Contents lists available at ScienceDirect

INTEGRATION, the VLSI journal

journal homepage: www.elsevier.com/locate/vlsi

On supporting rapid prototyping of embedded systems with reconfigurable architectures

I. Koutras*, K. Maragos, D. Diamantopoulos, K. Siozios, D. Soudris

School of Electrical and Computer Engineering, National Technical University of Athens, Zografou Campus, 15780 Athens, Greece

ARTICLE INFO

Keywords:
Reconfigurable architectures
Rapid prototyping
Genetic algorithm

ABSTRACT

Reducing time-to-market while improving product quality is a big challenge. This paper proposes a software-supported framework for rapid prototyping that offers a concurrent fast hardware/software system-level design. The introduced framework enables the constant evaluation and verification of the prototype under development, while it provides automatic functionality mapping to hardware via High-Level Synthesis techniques. We evaluate our framework and its software instantiation with a computer vision algorithm. Based on our experimentation, we show that our approach reduces the development time by almost 64×, it prunes the hardware design space by 34×, while maintaining designs that trade-off high Quality-of-Report on the Pareto frontier.

1. Introduction

Designing full system solutions is a complex task. With vastly increased complexity and functionality especially in the nanometer era, where hundreds of millions of transistors are integrated on a single chip, the design of complex Integrated Circuits (ICs) has become a challenging task. In addition to that, the continuously increased demand for even higher performance, i.e. in terms of operation frequency and power consumption, imposes that new design techniques are absolutely required.

This problem becomes far more important if we take into consideration that software aspects of ICs can account for 80%, or more, of embedded systems development cost, making the conventional way for product development insufficient. For instance, the International Technology Roadmap for Semiconductors (ITRS) [1] predicts that software development costs will increase and will reach rough parity with hardware costs, even with the advent of multi-core software development tools.

Electronic Design Automation (EDA) tools are crucial nowadays for deriving optimal solutions. Existing working flows are built on the fundamental premise that models are fully interchangeable and interoperable among different EDA vendors for the whole physical prototyping process, as in architectural analysis, simulation and synthesis. Even though this concept seems straightforward and promising, it has been proven completely elusive in the world of Electronic System Level: existing solutions do not provide either model interoperability, neither independence between model and software tools. As such, it is often desired to reach the highest possible systemic level of the target application description in order to avoid a possible vendor lock-in.

Apart from the technology-oriented parameters that affect the efficiency and/or the flexibility of a digital system, the tight time-to-market requirements make conventional ways for product development, e.g. start software development after finalising hardware, to lead usually in missed market windows and revenue opportunities. Hence, there is an absolute requirement for software developers to get an early start on their work, long before the Register-Transfer Level of the hardware is finalised.

Towards this direction, and as research pushes for better programming models for multi-processor and multi-core embedded systems, Virtual Platforms (VPs) solve one of today's biggest challenges in physical design: to enable sufficient software development, debug and validation before the hardware device becomes available. More specifically, with the virtualization feature, it is possible to model a hardware platform consisted of different processing cores, memories, peripherals, as well as interconnection schemes, in the form of a simulator. Furthermore, as the task of hardware development progressively proceeds, it is feasible to redistribute to software teams updated versions of the VP, that enable a gradually better description of the target architecture.

The concept of virtualization is also important for hardware architects, as it enables easier verification of Intellectual Properties (IP) kernels. This feature could be employed both in the case where only a few of the application kernels have to be developed in hardware, as well as if incremental system prototyping is performed. In both

* Corresponding author.
E-mail address: joko@microlab.ntua.gr (I. Koutras).

http://dx.doi.org/10.1016/j.vlsi.2017.02.007
Received 30 April 2015; Received in revised form 16 February 2017; Accepted 16 February 2017
Available online 16 March 2017

cases, the virtualization feature provides all the necessary mechanisms for performing co-simulation and verification between the IPs developed in Register Transfer Level (RTL) and the rest application functionalities executed onto the VP.

In this paper we identify common pitfalls during virtual prototyping for hardware/software co-design and propose a software-supported methodology to perform rapid system-level prototyping of complex digital systems. More specifically we:

- Present a modern virtual prototyping platform in Section 2 and explain some of the most time-consuming steps in development.
- Define and extract necessary task information through profiling, high-level synthesis and task execution on an FPGA (Section 3.1).
- Model task mapping as an optimisation problem and solve it with genetic algorithms (Subsection section 3.2).
- Optimise the hardware-assigned tasks assigned to hardware by performing further DSE with the help of FPGA (Subsection section 3.3).
- Evaluate our proposed working flow on the Harris & Stephens Corner Detection Algorithm from Computer Vision (Section 4). From a full-software solution we reach to an optimal mixed (hardware/software) one 6 times faster than a conventional approach to virtual prototyping (Section 5).
- Mention other relevant techniques that can be used for task partitioning, as well as other prototyping frameworks, and explain where and why our proposed toolflow works better (Section 6).

Our conclusion is that rapid prototyping of multi-million gate systems is achievable. We can have prototypes of our system on-the-go, as we modify, add, or optimise the algorithms that describe the system tasks.

2. Background: virtual prototyping

Fig. 1 depicts three consecutive design stages while using a virtual prototyping platform: (i) system modeling, (ii) rapid virtual prototyping and (iii) system integration. Different virtualization environments can be employed for this purpose. Without affecting the generality of VPs, we refer here to the OVP [2], since it is a publicly available and easily extensible approach. Additionally, the increased simulation speed provided by OVPSim ensures that complex systems can be modeled in reasonable amount of time (hundreds of millions of simulated instructions per second). As the OVP models are pre-built, they support fully functional simulation of a complete embedded system. Also, since these models are binary-compatible with the simulated hardware, the developed software can be executed onto the final system without any modifications. This enables faster iteration for the software development teams.

Similarly, hardware developers are also benefited from the adoption of hybrid VP. Since this platform is composed of a simulator and TLM/SystemC models, it exhibits increased flexibility which in turn alleviates many constraints that designers face during the architecture design. More specifically, the former models (related to OVP) describe the software part of the target system (e.g., executed onto an embedded processor), while the TLM/SystemC models provide the design functionality that has been mapped to custom hardware IPs (e.g., FPGA) after system mapping.

Connecting the hardware IPs with the functionality mapped to software is imperative in hardware/software co-design. Platform connectivity between the off-chip world and the functions in software is necessary once functions are implemented in hardware. Accordingly, a communication layer between the hardware-dependent software and the custom hardware IPs is necessary, as well as to provide all the necessary synchronisation for the computation tasks mapped in hardware and software. In the VP approach we refer to, this is implemented as a software stack running on a native host (x86-compatible) and more specifically to Fig. 1 as the "HotTalk API".

HotTalk API [3] provides a wide class of middleware stack, including device drivers for the host PC, libraries in OVP and transactors in FPGA, so that designers can efficiently test the entire system from early design iterations down to the final system validation with real-world test benches, with the minimum possible effort. The

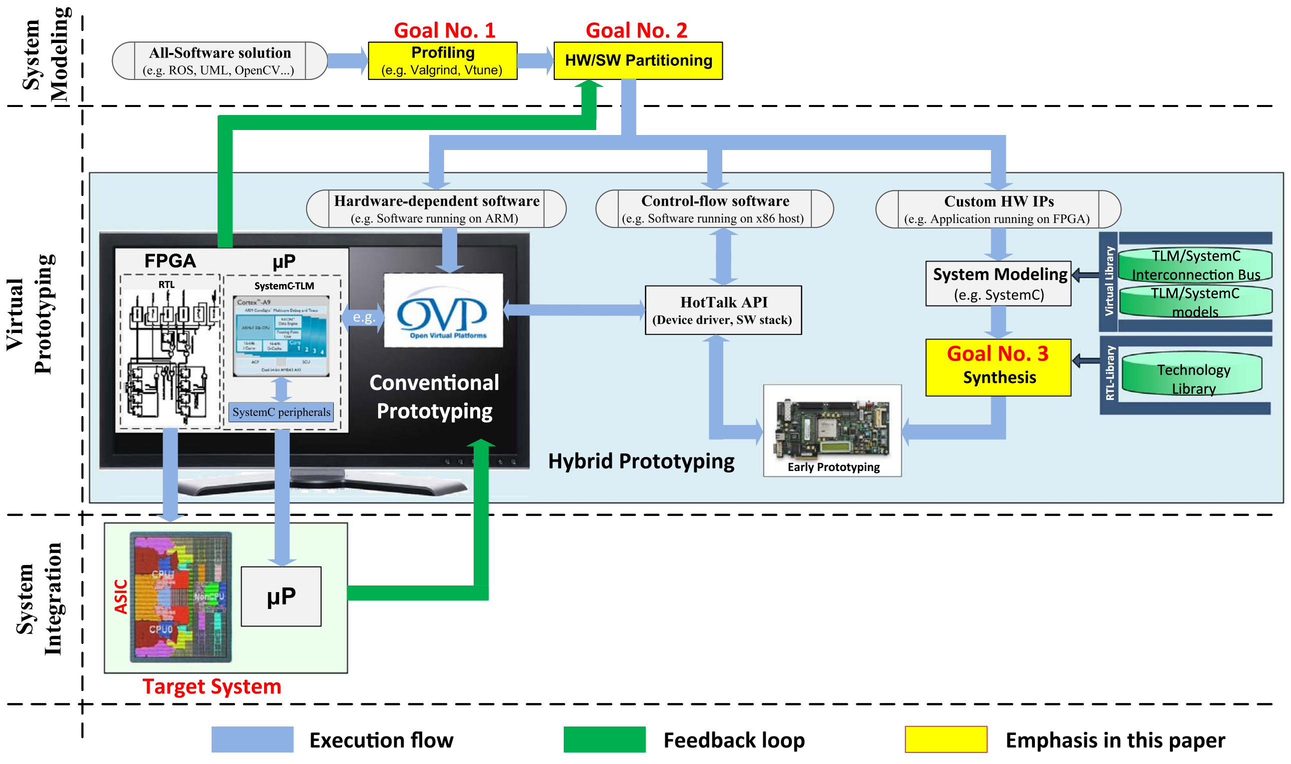

Fig. 1. A hybrid platform for virtual prototyping.

implementation of HotTalk API relies on existing physical communication interfaces (e.g., Ethernet or PCI Express). As mentioned previously, such an incremental design flow provides all the necessary information about meeting the system specifications, which in turn can be used for performing additional optimisations of the whole system, or re-partitioning the software through the feedback loop.

Having described the components of the virtual platform, the work flow becomes self-evident: We start from a high-level system modeling that meets the design specifications. This includes a software description of all the necessary application tasks, the application constraints and the constraints that the currently selected hardware platform (both in terms of the host CPU, memory and the FPGA). After having all the necessary information, the designers may proceed to the task mapping: they have to decide on which task runs where. A profiling step might precede this, to back up their decisions properly. Eventually we reach the stage of more hardware-dependent software development and hardware development. For the first part a target compiler might come in handy, for the latter High-Level Synthesis (HLS) tools could speed up the process by using the software code directly instead of rewriting it in a Hardware Description Language (HDL). The design flow we describe supports constant on-going progress: As long as new IPs are developed, the hardware design team is able to incrementally test these IPs by replacing a functionality of the employed C/C++/SystemC/TLM model with the equivalent HDL prototype mapped onto FPGA boards.

Even though the presented design flow seems complete and there are plenty of design tools that tackle software and hardware problems individually, there are only a few approaches that leverage problems arising in systems that tightly integrate software with custom hardware. This occurs mainly due to the system integration challenges that have to be addressed. Even limited, there are EDA approaches which promise to alleviate the integration problem in RTL simulation, emulation and prototyping environments. However, these solutions are often too complex, slow and expensive: frameworks as Corsair [4] require high-level specifications in languages such as PMSC and SDL, and multiple FPGAs for the hardware implementation. Moreover, in such devices the communication link between the host computer and the prototyping hardware is the most constrained [3].

In any case, the burden of efficient task mapping relies solely to the system designers. Even if the virtual prototyping allows fast and reliable estimation of a solution, the designers have yet to figure out which solutions from a vast design space they need to evaluate. Choosing in which side, software or hardware, a task should be implemented and run has a certain impact to the performance and the resource usage of the application. Tweaking the software compilers and the HLS compilers may also lead to different results. Designers should take into consideration all these possible options they have in design before even reaching to a viable solution in the VP.

3. Proposed rapid prototyping framework

Towards to the direction of using more effectively the virtual platform designers have in their disposal, we propose a software-supported framework for enabling product development jointly by software and hardware teams in a way that close interaction is allowed during the development phases. The main competitive advantages of this framework are automatic partitioning of system functions to hardware and provided PC-based co-simulation, which trade-offs between speed (functional simulation) and accuracy (cycle-accurate simulation) depending on design requirements. Finally, the supported feature of HLS enables designers to perform more sufficient exploration of the design space (candidate architectural solutions) without constraints related to the time-to-market pressure.

The introduced framework also addresses limitations posed during the profiling step. More specifically, in case where the profiling procedure is performed on a different platform from the actual target system, this might lead to inaccuracies in the derived conclusions. For instance a different platform imposes changes in the Instruction Set Architecture, the microarchitecture and the compiler, resulting in variances in the executable that is profiled.

In our framework we define three development steps: (i) Task Characterisation (Goal No. 1), (ii) Application Mapping Exploration using a Genetic Algorithm (Goal No. 2), (iii) Hardware tasks synthesis optimisation using an FPGA (FPGA-in-the-loop Design Space Exploration) (Goal No. 3). Up to now, these steps (depicted also in in Fig. 1) were performed manually, whereas throughout this paper we propose systematic methodologies in order to support them.

After completing the last stage of the design flow, system integration is possible. During this stage, the different cores of target system, including among others the embedded CPU, the reconfigurable fabric, as well as the memory components are integrated to form the target system. Towards this goal, our framework employs an approach for quantifying with acceptable fidelity the potential gains of the system implementation in terms of maximum operation frequency, amount of utilised resources, communication overhead between embedded CPU and reconfigurable platform, etc.

Next subsections give additional technical details about the employed steps in our framework.

3.1. *Task characterisation in software and hardware*

As presented in Section 2, system modeling describes the stage where the development team provides an abstract description of the system architecture and starts planning the way that this functionality has to be modeled into hardware and software kernels. All the top-level constraints have to be met, but since this is an early design step, the functionality of the system is still described with an “all-software” solution.

Starting from an all-software solution, the characterisation step begins. We profile initially the application algorithms to determine those kernels that highly affect the system performance. Different criteria might be incorporated for this task, while the most common ones affect the determination of computationally intensive tasks, the tasks with increased demand for I/O communication, as well as those tasks that can be executed much faster if we extract their inherent parallelism. For this purpose, a number of software tools are employed: Callgrind [5] to extract the complete dependency graph of function blocks which are eventually the system tasks, Valgrind [6] to estimate the memory usage of the function and VTune to extract the amount of instructions each function block requires. Furthermore, we record the size of the input and the output for each function block.

As for the possible hardware mappings, we need to perform an initial HLS of the candidate function blocks for the target architecture (target FPGA device). Fine-tuning of the HLS process is not required at this point; we will perform a design space exploration for the possible HLS configuration settings in a later step (Subsection 3.3), once the mapping of the tasks to the software or the hardware side will have been performed.

It is true that the most accurate profiling information is available only after the first design prototype is developed. Typically, once a project has reached this stage, most of the budget has been sunk, which is usually beyond the point of “no return” [7]. In contrast, the proposed framework offers the flexibility of incremental hardware development and system testing under real world constraints, so that accurate profiling information can also be provided through development stage with a feedback loop. During these possible iterations, this initial step reflects the most up-to-date profiling information, so that the following step, the mapping, does the task partitioning with correct criteria.

3.2. *Application mapping exploration using GA*

Based on the results derived from the Task Characterisation (profiling), the next step regards the automatic exploration of the

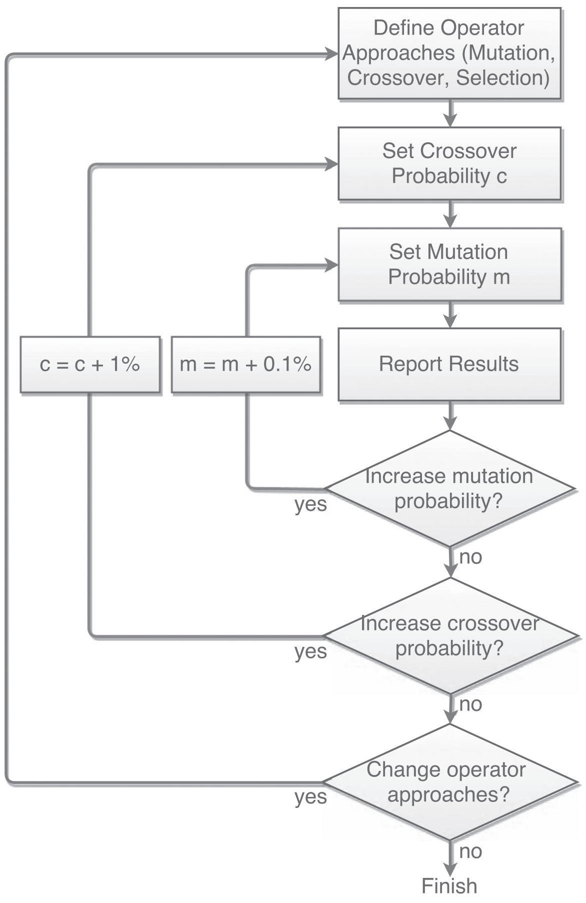
Define Operator Approaches (Mutation, Crossover, Selection)
Set Crossover Probability c
Set Mutation Probability m
c = c + 1%
m = m + 0.1%
Report Results
Increase mutation probability?
yes
no
Increase crossover probability?
yes
no
Change operator approaches?
yes
no
Finish

Fig. 2. Calibration scheme of genetic algorithm.

application mapping on software/hardware. In this step we perform the classification of the initial "all-software" solution to two possible categories: (i) software tasks executed on the Host CPU, and (ii) hardware tasks implemented as hardware IPs (accelerators) on the FPGA.

Typically, this step implies a graph partitioning problem between software and hardware, where we target at the optimisation of more than one objectives under the consideration of specific constraints. Graph partitioning is known to be a NP-hard problem. A widely used approach to encounter this problem is by using heuristic and approximation methods [8]. In this work we choose the employment of a genetic algorithm to address the partitioning problem and provide considerable quality of solutions in a fast manner. Our approach relies on the NSGA-II [9] which is a state of the art multi-objective genetic algorithm. NSGA-II has been widely used in many classes of problems [10]. Moreover, NSGA-II can be found available as open-source software which allows us to customise the algorithm and adapt it to the target problem accordingly .

Genetic algorithms are based on an iterative procedure applied to a population of *individuals*, where each of the individuals represents a candidate solution called as *chromosome*. A chromosome is encoded by a finite string of symbols from a given alphabet, known as *genes*. Considering the formulation of our problem, a gene defines the partition where a module (e.g. system task) is assigned to. In Fig. 3 the formulation of a chromosome consisting of seven genes is illustrated, where each gene represents a system task that has been assigned in either software (A) or hardware (B). During the execution of a genetic algorithm, in each iteration the chromosomes receive a score (*fitness*) which quantifies their efficiency and a number of genetic operations, namely *selection*, *crossover* and *mutation*, are applied to them in order to produce the next generation of advanced chromosomes (next iteration).

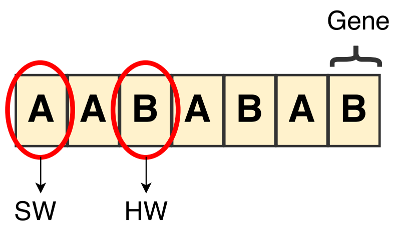

Fig. 3. Formulation of chromosome. Each gene denotes a task that has been assigned either to software or hardware.

In the current paper we customised NSGA-II and experimented with various widely-used operator methods to investigate the most appropriate configuration which leads to the best quality of solutions. These operator methods are supported by our algorithm and can be selected at compile-time. More specifically, for crossover operation we considered three different methods; one-point, two-point and uniform. Similarly, for the selection operation we considered the roulette wheel, tournament and the rank method. Regarding the operation of mutation, we experimented with the uniform (in our case similar to bit inversion method as two distinct partitions, software and hardware, take place) and the order changing method. Moreover, to enhance the efficiency of our genetic algorithm we followed a calibration scheme to explore a plethora of operator combinations and fine-tune the relative parameters. Our calibration scheme is based on multiple-level loops implying a top-down tuning approach illustrated in Fig. 2. The outermost level loop refers to the selection of the operators approaches. The mid-level loop regards the set of the crossover probability and the innermost the definition of the mutation probability. The tuning of crossover and mutation probabilities is performed gradually by increasing the relative parameters in each iteration. After an exhaustive exploration we concluded to a combination of fine-tuned operators that lead to the best performance. These operators are described in the following paragraphs.

Initially, the proposed algorithm begins with the initialisation phase, where a population of chromosomes with random genes is generated. The variety of the randomly generated chromosomes results to a good spread of solutions which guarantees the expansion of the solution space. Thereafter, based on this initial population of chromosomes the basic iterative operation of the genetic algorithm takes place to generate optimised solutions for the partitioning problem.

Selection. The operation deals with the selection of chromosomes from the current population for breeding new offsprings. The selected chromosomes are referred as *parents* for the upcoming generation. The method we chose for this operation relies on a tournament selection algorithm, where two candidate parents are randomly selected and compared. The winner (parent) is obtained according to NSGA-II sorting approach which analyzed in detail in [9]. This sorting approach implies that each chromosome is characterized by two metrics; the nondomination rank and the crowding distance rank. These metrics values are extracted by the position of chromosomes in the nondominated fronts (including Pareto-optimal front). The procedure of the applied operation is illustrated in Fig. 4.

Crossover. The crossover operation deals with the generation of new offsprings. Our algorithm employs the one-point crossover approach, as it is depicted in Fig. 5. Initially, a randomly selected cross-point is computed; then, two new offsprings are generated by combining the parent's genes with reference to the selected cross-point. The randomness of the employed crossover minimises the probability to be trapped to local minima. According to our experimental exploration, we found that the most efficient partitioning results are retrieved when the probability *c* of applying crossover operation was around 70–90%.

Mutation. The mutation operator maintains the diversity of chromosomes through the generations, by altering the offspring's genes.

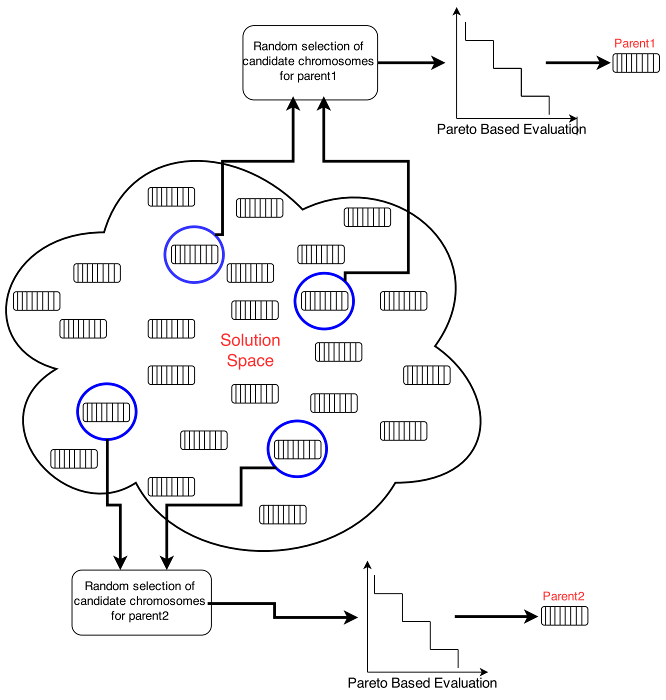
Random selection of candidate chromosomes for parent1
Pareto Based Evaluation
Parent1
Solution Space
Random selection of candidate chromosomes for parent2
Pareto Based Evaluation
Parent2

Fig. 4. The process of parent's selection for breeding new offsprings.

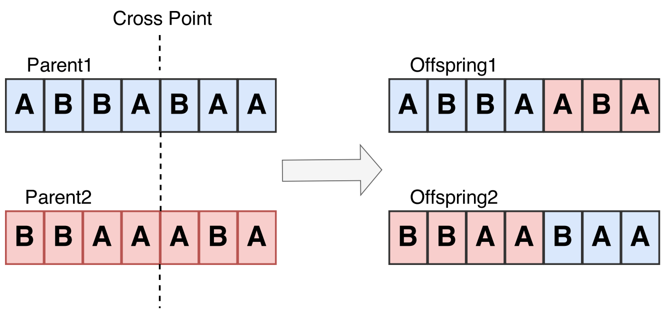

Fig. 5. One-point crossover approach. The offsprings result from the parent's genes combination.

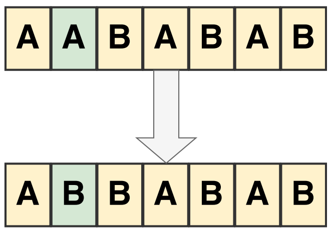

Fig. 6. Uniform mutation approach. Each node has a small probability to be assigned into a different partition.

The employed mutation relies on the uniform approach as shown in Fig. 6. In this operation there is a uniform small probability for each node to be moved into a different partition. Based on our experiments we found that a good compromise between the quality of solutions and the avoidance to be trapped to local minima is achieved for probability m 0.5–2%.

Objectives & Constraints. Towards adapting NSGA-II in the software/hardware partitioning problem, we customised the algorithm in a sophisticated manner. We defined specific objectives which seek to optimise various aspects of the system that highly affect the overall performance. More specifically, our optimisation goals imply high quality partitions considering both the execution speed and the utilisation of communication link between the software and the underlying hardware platform. The employed objectives are summarised as follows:

- *Objective*$_1$: Minimise the total execution delay of the system. This objective is based on the hardware and software execution delay of each task separately and the delay due to the data transfers between the hardware IPs and the host PC. Thus, we seek to optimise the total delay according to the following formula:

$$totalDelay = \sum_{i=1}^{M} \mathrm{SwDelay}(i)\mathrm{SwCalls}(i) + \sum_{j=1}^{N} (\mathrm{HwDelay}(j) + \mathrm{CommOverhead}(j))\mathrm{HwCalls}$$

  where M indicates the number of software assigned tasks and N the number of hardware assigned tasks.
- *Objective*$_2$: Minimise the number of interactions between hardware and software tasks. This objective implies minimising the communication overhead (data transfers) between the host platform (embedded CPU) and the FPGA.

Furthermore, in order to provide appropriate solutions that match the capabilities of the underlying hardware, we employed a set of constraints that aim to give a biased direction to the exploration of the solution space in function of the target platform (Host PC and FPGA) characteristics. These characteristics mainly refer to the available resources of the system: LUTs, RAMBs, DSPs, I/O blocks regarding the FPGA device and memory requirement for the Host PC. Overall, the proposed, customised NSGA-II supports platform-oriented solutions by investigating the system's full potential to provide tailored solutions in accordance with the system specifications.

3.3. FPGA-in-the-loop design space exploration

Co-designing a system simultaneously on software and hardware level may lead to under-designed system implementations not meeting all non-functional properties such as timing, cost, or power consumption. Alternatively, design decisions for the allocation of resources might have been taken too strictly, leading to too costly, due to Engineering Change Orders (ECO), and thus over-designed system implementations which may reduce the later win margins per unit sold. Consequently the design complexity in such systems makes the Design Space Exploration (DSE) a distinguishing element of HW/SW co-design technology.

Prior art [11,12] provides a wide field of DSE techniques in chip design, ranging from single exploration cost algorithms, i.e. randomised search techniques, techniques relying on iterative improvement such as simulated annealing, or exact techniques based on integer-linear program (ILP) formulations, to complex multi-objective optimisation algorithms, i.e. Pareto-Front exploring evolutionary algorithms such as SPEA2 and NSGA-II. However, during the evaluation steps a high number of design solutions must be implemented. While this step may be of low implementation effort in software, i.e. changing some variables' values, it is a tough task in hardware design.

We propose to fasten the DSE process by incorporating an iterative high level synthesis process of the under-development hardware IPs. The rest system, apart of these IPs, relies on virtual modules so that the overall simulation runs fast enough. While the designer is working only in software layers, it is highly possible to decrease DSE designs' evaluation time. Fig. 7 illustrates the proposed DSE framework. The application's control flow graph is composed of computation kernels that shall be mapped to FPGA in order to be accelerated or stay on host CPU, according to their impact on system's performance, i.e. execution time, memory transfer cost, IO activity etc. Given the target co-design architecture, a high number of mapping configurations may be setup. For example, one node of applications' CFG shall be mapped to FPGA, while the others stay on CPU. In order to quantify the performance of this system's configuration, we employ HLS tool to rapid map this CFG

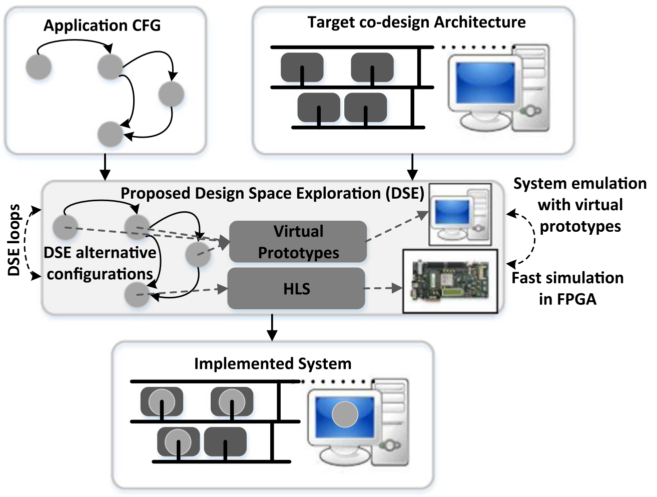
Application CFG
Target co-design Architecture
Proposed Design Space Exploration (DSE)
System emulation with virtual prototypes
DSE loops
DSE alternative configurations
Virtual Prototypes
HLS
Fast simulation in FPGA
Implemented System

Fig. 7. Proposed FPGA-in-the-loop Design Space Exploration.

node to RTL and execute it on real FPGA device. The rest application is executed as software through virtual prototypes that interact with the FPGA using appropriate API (HotTalk). This early-prototyped system is executed fast, since the host CPU-mapped nodes are emulated on high-speed CPU, while the FPGA-mapped CFG nodes are simulated in real-time onto the FPGA.

From the different configurations of applications' CFG partition, the designer should find the one that best fits to required performance metrics. However, given that a complex application may contain tens to thousands of CFG nodes and a modern HLS tool provide a number of architecture optimisations, the design space may become too big to be explored rapidly. For this scope we propose a systematic classifier for HLS optimisation techniques, which provides the appropriate rules for applying every technique according to system's desired metrics. Typical examples of the proposed classifier for the Xilinx Vivado HLS tool [13], which is used throughout this study, are illustrated in Fig. 8 and Fig. 9. The first one depicts the classification of synthesis directives that are applied to application's source code into three main classes, regarding the optimisation criterion, i.e. latency, throughput and area. The second one classifies the throughput-related directives.

The iterative DSE exploration process is illustrated in Fig. 7. The goal of this process is to automatically map computational kernels of the input applications written in C/C++/SystemC to a heterogeneous MPSoC platform. By automating as many design steps as possible, an early evaluation of different design options is possible. Different hardware accelerators may be generated automatically for nodes in the application's CFG. The designer defines an MPSoC system model from resources in the vendor library. This library contains synthesizable IP cores such as DSPs, buses, memories etc. During DSE, allocations as well as bindings of tasks to processing resources and communications to routes in the architecture are determined and evaluated for objectives such as latency, throughput, power, and area using the concept of classified HLS directives. The derived solutions are evaluated using FPGA-in-the-loop system prototyping. A special module of our framework is responsible to prune some configurations from design space, so that the overall evaluation time is further decreased, compared to a brute-force exploration technique. The module implements simple architecture-based rules in order to provide the pruned component design space, e.g. the pruning of pipelining technique when the maximum area constraint is reached. At the last iteration step, the designer may then select promising implementations for subsequent optimal system setup.

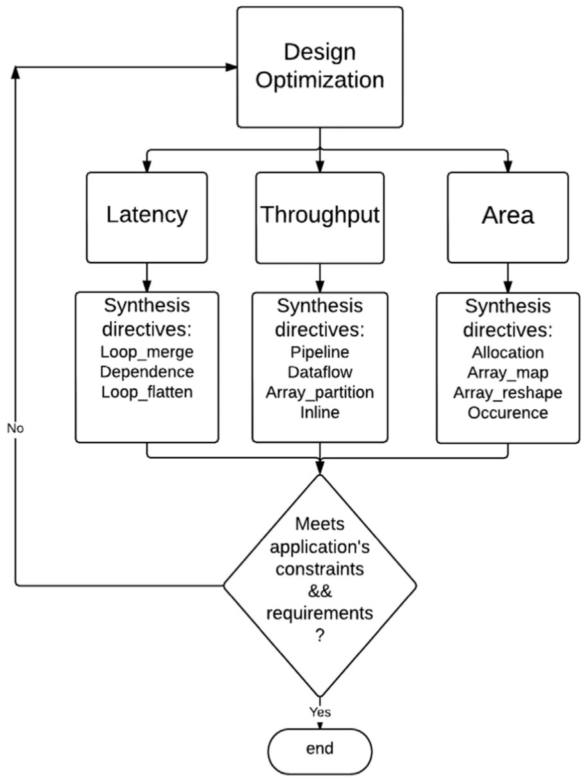

Fig. 8. Classification of Vivado HLS directives regarding the optimisation criterion, i.e. latency, throughput and area.

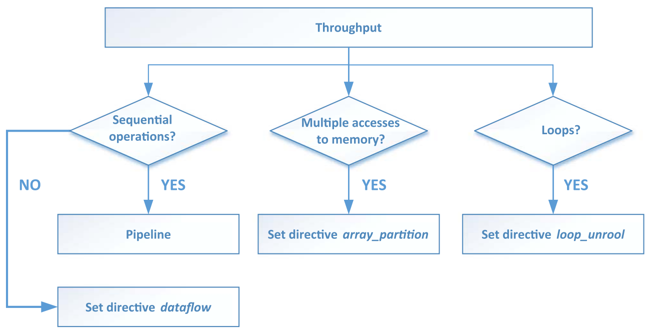
Throughput
Sequential operations?
Multiple accesses to memory?
Loops?
NO
YES
YES
YES
Pipeline
Set directive array_partition
Set directive loop_unrool
Set directive dataflow

Fig. 9. Throughput-related directives classification for Vivado HLS.

4. Evaluation system setup

This section introduces the use case employed for demonstrating the efficiency of our rapid prototyping framework. Towards this direction, initially we discuss the limitation posed by the implementation of Computer Vision (CV) algorithms and then we emphasise on the target limitations posed by the efficient implementation of a representative CV algorithm, namely the Harris & Stephens Corner Detection Algorithm.

Use case: computer vision algorithm

The complexity of CV algorithms combined with the growing demand for applications with intensive amount of information (like high definition images or videos) leads to greater amount of computational power. This complexity problem becomes far more savage by taking into consideration that CV algorithms often demand for example, nonlinear optimisations, in order to be more accurate. In

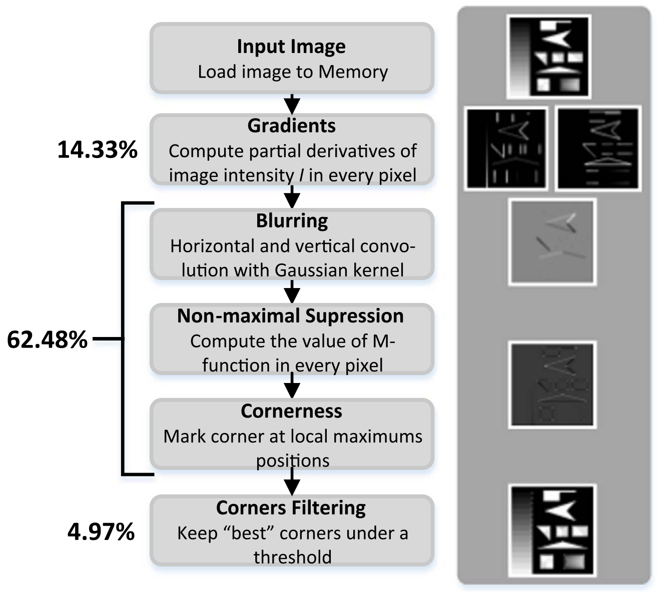
Input Image
Load image to Memory
Gradients
Compute partial derivatives of image intensity I in every pixel
14.33%
Blurring
Horizontal and vertical convolution with Gaussian kernel
Non-maximal Supression
Compute the value of M-function in every pixel
62.48%
Cornerness
Mark corner at local maximums positions
Corners Filtering
Keep "best" corners under a threshold
4.97%

Fig. 10. Profiling and configuration of the Harris & Stephens corner detection algorithm.

addition, concerning image processing, the computational cost for ordinary image sizes ranging from 640×480 pixels (VGA) to 4096×2160 (4 K) can reach, sometimes, the level of several million operations, while requesting memory allocation of several gigabytes of RAM. So, their implementation cannot be considered as efficient with the usage of general purpose CPU's, which are sufficient exclusively for those systems that exhibit low complexity (e.g., small image size, low video quality and low frame rate). On the contrary, for more demanding requirements specific processors like GPUs perform better [14,15].

One of the main targets of the field of CV algorithms is to extract features from images and supply the results as inputs to systems, in order to make decisions (such as triggering an actuator to move a robot). This process is called feature detection and refers to all methods and operations that are necessary to calculate at every pixel of an image whether or not satisfies the criteria of each feature. The result is a subset of the image, containing either isolated points, continuous lines or connected regions. Although there is not a clear definition of the meaning of feature, usually we refer to feature as an interesting part of an image, which is repeated two or more times throughout the image.

In order to demonstrate the efficiency of our introduced framework, this paper discusses the implementation of Harris & Stephens corner detection algorithm [16]. The control flow graph of Harris & Stephen algorithm is depicted in Fig. 10. The algorithm is a data streaming application, meaning that the input image is inserted on the first computation node and then, by several sequential computations, the output image is annotated with the edged corners found. Previous analysis highlighted the efficiency of this algorithms as compared to alternative implementation for feature extraction (as in SURF) [17]. Thus, one can claim that such an algorithm can benefit for the usage of massively parallel computational platform, such as the reconfigurable architectures. More specifically, the target implementation medium consists of a recent Xilinx Kintex-7 FPGA board (xc7k325tffg900-2), as well as an embedded CPU. The role of embedded CPU is to perform mainly the control tasks of the entire system and the task(s) involving floating point arithmetics.

The implementation of CV algorithm exhibits a number of challenges that should be sufficiently addressed for an efficient co-design. These challenges affect both architectural and algorithmic parameters. Among others, the proper communication scheme between the reconfigurable architecture and the embedded processor highly affects the performance metrics, since it can stall the system execution. For this purpose, different parameters that affect the communication scheme (e.g., packet size, arbiter mechanism) must be well studied and quantified. Also, we have to mention that the communication scheme is tightly firmed to the output from the H/W-S/W partitioning, as well as to the number of times each task is invoked.

5. Experimental results

This section describes the experimental results by applying the introduced framework for implementing the Harris algorithm onto a recent Xilinx Kintex-7 (xc7k325tffg900-2) board. Initially, we profile the Harris algorithm in order to determine the most computational intensive tasks. For this purpose the Valgrind suite [18] was employed. The results of this analysis are depicted, as percentages over total execution time, on the left part of Fig. 10. Based on this analysis, we can concentrate our efforts in order to improve performance metrics. Specifically, the majority of optimisations should be applied to ”imgblurg“ function, since it occupies almost 62% of the total execution cycles. Additionally, in a HW/SW co-design approach, this function has to be implemented onto HW (e.g. FPGA) because such a selection will accelerate significantly the system's maximum operation frequency.

5.1. Optimisation phase for Vivado HLS

This subsection shows the obtained results for various optimisations that can be performed using the features provided by Vivado HLS tool, in order to further improve design metrics. By default, Vivado HLS tries to create the most optimal implementation, according to the design's requirements. The clock is the first constraint to be determined and Vivado HLS uses the specification of the target device to determine the maximum number of operations that can be executed, within a clock cycle. After achieving the optimum clock frequency, Vivado HLS produces the synthesised circuit and makes optimisations automatically according to these goals:

- Improve throughput
- Reduce latency
- Minimise area (resource) requirements

Table 1 summarises the Vivado HLS directives that were employed throughout this study.

In addition to the default synthesis operations, we incorporate a number of synthesis directives and configurations which optimise the efficiency of the previously mentioned three design parameters, depending on inherent application's requirements. The first challenge towards this goal is the high memory allocation required by the

Table 1
Employed optimisation directives on Vivado HLS.

Optimisation	Directive	Description
Throughput	Pipeline	Cascading task assignments
	Dataflow	Changing sequential order to concurrent
	Array partition	Splitting the array into multiple smaller arrays
	Loop unrolling	Parallelising computations inside loops
Latency	Latency	Providing min-max margins for loops/ functions
	Loop merge	Merging loops with same bounds
	Loop flatten	Flattening nested loops
Area	Data types	Transforming built-in types to arbitrary precision
	Function Inlining	Replicating shared logic to private
	Array map	Combining small arrays into a larger one
	Resource	Guiding specific HW for source logic operators

algorithm. Since our target is an FPGA board, it is apparent that HLS compiler should be aware at compile time about the application's exact memory requirements. In order to optimise the memory footprint so that the algorithm could be synthesizable, we proceed to the following source-code modifications:

- We replaced the `malloc()` and `free()` calls, that are used for dynamic memory allocation with static ones. Arrays have been used as buffers wherever necessary. However, such a selection introduces additional memory requirements. Note that the constraint for memory savings is important in the FPGA domain due to limited on-chip available storage.
- We incorporated arbitrary precision data types, offered by Vivado HLS. By carefully examining the variables precision necessities, we provided fixed-point replacements of the built-in C data types (e.g. *apint21* requires only 21 bits instead of 32 bits of *int*)
- We applied parametric fragmentation of input image. Since the algorithm employs sequential computation on pixels, we divided the input image in bands, so that to process each band at a time in the FPGA and hence decrease memory footprint.

The aforementioned techniques are employed in order to address the memory requirements regarding to HLS implementation and further optimise the memory footprint. The implementation results are evaluated based on the usage of FPGA resources and timing.

Regarding resources, there are four different resource block types found on most FPGA devices: (i) block random-access memory (BRAM), (ii) digital signal processing (DSP), registers and (iv) look-up table (LUT). The results for our use case were taken after exploring the optimal configuration vector, through the proposed DSE methodology, and are summarised with both absolute values and relative ones to the total values of the target board in Table 2. It should be noted that the initial synthesis of the code is not implementable, since our first modification, the replacement of dynamic memory allocations with static declarations, results in 108% BRAM utilisation. The second technique of data types replacement of built-in data types with arbitrary precision data types reduces the memory by 1.9×. Finally, dividing the image input to bands for iterative processes results in 1.7×, 2.5×, 5×, 9.3×further BRAMs decrease. The usage of the other resource types seem to remain unaffected from our source-code modifications.

With regards to the latency and throughput optimisation stages, we applied several optimisation directives during HLS, as described in Table 1. An example of timing results for different directives one could take for the last synthesis of Table 2 ("Input fragmentation/16″) is found in Table 3. While the *Pipeline* technique seems to provide the best performance improvements in terms of latency, however it highly increases the resources utilisation. Thus, in order to find the best system configuration, the designer should examine the trade-offs among all three metric spaces, i.e. area, latency and throughput. Such an analysis is described in the following section.

Table 2
Memory-optimised synthesis results of Harris algorithm using Vivado HLS (Kintex7-xc7k325tffg900-2).

Optimisation phase	BRAMs		DSPs		Registers		LUTs	
	Abs.	%	Abs.	%	Abs.	%	Abs.	%
Initial (no malloc/free)	964	108	111	13	11,534	2	24,350	11
Data types optimisation	499	56	69	8	10,692	2	23,350	11
Input fragmentation/2	301	33	76	9	11,399	2	24,652	12
Input fragmentation/4	202	22	76	9	11,324	2	24,584	12
Input fragmentation/8	103	11	76	9	11,420	2	24,565	12
Input fragmentation/16	56	6	76	9	11,431	2	24,537	12

Table 3
Timing-optimised synthesis results (ms) of Harris algorithm using Vivado HLS (Kintex7-xc7k325tffg900-2).

Image size	128×128	256×256	1024×1024
No-opt.	25.7	101.3	24,398
Loop flatten	24.3	95.6	23,022.2
Pipeline	12.9	50.8	3013.7
Pipeline+Array partition	13.4	52.7	3225.9
Array partition (*no*-Pipeline)	27.8	109.6	6700.1
Loop unroll–10	24.2	95.3	5824.2
Loop unroll–50	24.1	94.9	5798.2

5.2. Rapid prototyping phase

This subsection shows the effectiveness of the combined FPGA-in-the-loop with DSE through HLS flow. Having selected the last solution of Table 2, we created a configuration vector of combined optimisation techniques from Table 1, which formulate a design space of 3052 solutions, i.e. all the possible design solutions from combinations of throughput-latency-area optimisation techniques in Table 1. We specifically chose such a small setup so that we could evaluate all those solutions with and without the DSE pruning technique. We found that by appropriate guiding this step we could extract for evaluation only 88 of them, since the rest gradually lead to worse metrics. After evaluating all 3052 solutions we found that every one of the 88 DSE-selected solutions was on the Pareto front. Thus, the overall evaluation time was decreased by 34×. The results are depicted in Fig. 11. The XYZ-axes depict the normalised throughput, area and operating frequency compared to the highest measured respective metric. The Pareto front among the three optimisation targets is formulated by solutions that provide a) small area footprint with high frequency but low throughput due to latency and area optimisation directives, i.e. *latency, loop merge, loop flatten and data types replacement* and b) high throughput and frequency but increased area, due to directives that provide replicated logic in order to decrease shared resources and thus increase throughput, i.e. *inlining, loop unroll, dataflow, pipeline and array partition*.

Finally we evaluate the effectiveness of the employed FPGA-in-the-loop DSE. We measured the overall elapsed simulation time, the RTL

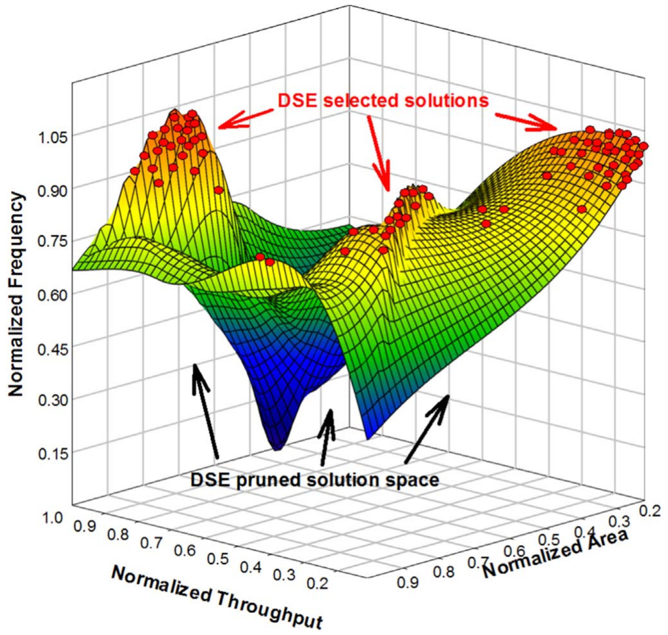

Fig. 11. Effectiveness of the proposed DSE-pruning flow: The framework evaluated 88 solutions from a total of 3052 of the design space, all lying on the Pareto front.

Table 4
Qualitative comparison among tool flows for prototyping platforms.

Simulation method	RTL simulation cycles	Simulation Time (sec)	IPS
RTL (ISIM)	141,848,837	10,894	13,020
FPGA-in-the-loop	4,136,162	1703	83,293
Speed-up	**34.3**×	**6.3**×	**6.4**×

simulation cycles (i.e. while executing solely on Xilinx ISIM [13]) and the obtained instructions per second (IPS). We measured two evaluation tests for the Harris algorithm using an input image of 1024×1024, one using only RTL simulation of the entire system and the other using the FPGA device, having porting the *imgblurg* function (almost 65% of the total execution time), while the rest logic run on Host CPU. Table 4 summarises these results. We found that the proposed prototyping framework is able to deliver a simulation environment of up to 6.3×faster, which translates to an increase of IPS at around 6.4×. This is achieved due to the fast simulation of computational intensive *imgblurg* function compared to slow emulation on Host CPU, i.e. the case of solely RTL simulation. Such high speed-up gains increase significantly the overall DSE exploration.

Consequently, on our exploration regarding Harris algorithm, the time of design exploration was decreased by a) 34×due to DSE pruning and b) further 6.3×due to the FPGA-in-the-loop prototyping.

6. Related work

The partitioning problem constitutes an internal part of the modern system design, due to the continuously increasing needs for higher performance that exist simultaneously with the resource utilisation bottlenecks. Numerous algorithms in literature have been proposed to address this problem, although no known polynomial-time, globally optimal algorithm for balance-constraint partitioning has been suggested so far. However, several efficient heuristics were developed which find high-quality circuit partitioning solutions, a similar type of partitioning problems, and in practice are implemented to run in low-order polynomial time [19]. The Kernighan-Lin (KL) algorithm [20] performs partitioning through iterative improvement steps. Despite the fact that it does not always find an optimal solution, the KL algorithm performs reasonably well in practice on graphs with up to several hundred vertices. On contrary, the Fiduccia-Mattheyses (FM) [21] is a partitioning heuristic applied to netlists encoded in hypergraph form, which offers substantial improvements over the KL algorithm. The drawback of FM algorithm affects the quality of solutions retrieved for large designs. In such a case, the FM algorithm may terminate with a high net cut or make a large number of passes, each producing minimal improvement [19]. In order to overcome this limitation, the FM algorithm is usually embedded into a multilevel framework that consists of several distinct steps. We find those solutions inefficient for our partitioning problem; instead we use a genetic algorithm approach [22] to handle the partitioning problem. The evolutionary heuristic nature of this algorithm is suitable for performing an effective search space exploration, while it supports lexicography optimisation of many objectives simultaneously without taking into consideration only local information.

Prior art investigated the improvements of incorporating virtual prototypes for fastening time-to-market while offering high design quality-of-report. Authors in [23] present a survey of such technologies with major achievements of two decades of research on methods and tools for hardware/software co-design. They start with a historical survey of its roots, by highlighting its major research directions and achievements until today, and finally, by predicting in which direction research in co-design might evolve in the decades to come. They state that emerging methodologies such as virtual prototyping, co-simulation, and design space exploration has meanwhile shown indispensable benefits for much better optimisation results due to an early trade-off analysis and shorter time-to-market frame due to concurrent development of hardware and software. In [24], authors present a compositional methodology for HLS-driven design-space exploration of SoCs. Their approach is based on two novel algorithms that combine allow for deriving a set of alternative Pareto-optimal implementations for a given SoC specification by selecting and combining the best implementations of its components from a pre-designed soft-IP library. They operate at a higher level of abstraction, thus they can handle components specified in SystemC, which enables the parallelisation of HLS runs to construct system Pareto fronts.

Remarkable industrial tools have been recently presented, offering prototyping solutions. Synopsys Virtualizer [25] can exploit HAPS (High-performance ASIC Prototyping Systems) [26], a multi-FPGA used for physical prototyping, and used for rapid prototyping. Our methodology could use it, provided that the HotTalk API was implemented there and there was a way to report the application mapping exploration metrics. Likewise, Xilinx offers a co-simulation environment called Vivado System Generator for DSP [27]. The integrated development environment seems comfortable enough to develop a hardware/software prototype, but developers should port their software code to SimuLink which increases the development cost, compared to our approach.

Altera presented a simulator that enables early software development by emulating the ARM Cortex-A9 CPU(s) and the peripherals which are included into the Altera SoC FPGA devices [28]. In order to support hybrid prototyping, the simulator is connected with the host PC through a driver-based communication mechanism. Even though [28] supports fast functional simulation, the low timing accuracy (since it uses only functional simulation), introduces limitations to detailed software validation. Also a typical interface for hybrid prototyping is the SCE-MI solution from Accellera [29], which provides a communication interface between the host PC and an HDL simulator. In more detail, the host PC at SCE-MI executes a testbench software for manipulating the interface. To the other side, the system's IP kernels developed in HDL include all the necessary mechanisms for accepting the stimuli through the interface and send back to the host the appropriate data.

Authors in [30] developed a QEMU x86 emulator which is combined with a SystemC model. Although SystemC ensures the universality of this solution, as well as the unlimited support for CPU cores in the VP side, there are performance limitations related to the software execution on the x86 processor since it is actually emulated.

Our previous work [3] in this field includes a novel framework for supporting rapid, as well as incremental prototyping, of heterogeneous 2-D and 3-D embedded systems. The Plug & Chip framework provides to designer teams the desired connectivity between the hardware-depended software, the control-flow software, as well as the custom hardware IPs. Such a feature enables starting the development, testing and validation of the embedded software substantially earlier than it has been possible in the past. However, the framework does not target FPGA devices, while it does not deal with the effective DSE technique presented in this paper.

Finally, a work on many-accelerator systems and their respective rapid virtual prototyping is presented in [31], where the authors give an overview of how to explore effectively the different options in terms of computation and communication for such systems. Models need, however, to be developed in SystemC and the workflow is not speeded up via an FPGA.

7. Conclusion

A design framework for supporting rapid prototyping of multi-million gate systems was introduced. The framework efficiently eliminates the solution space by performing design space exploration pruning, while the under-evaluation design configurations are rapidly developed with the usage of HLS techniques. Then, the designs are evaluated using an early system prototype that is based on a host CPU and an FPGA device. The simulation on such a system is highly accelerated by the FPGA-in-the-loop module. The efficiency of this framework was evaluated by applying it to a computer vision algorithm with increased memory requirements. Experimental results show that our solution achieved to decrease the development time by almost 64×by pruning the HW design space by 34×, thus eliminating the design configurations that have to be evaluated, while maintaining designs that trade-off high Quality-of-Report (QoR) on the Pareto frontier.

References

[1] ITRS, International Technology Roadmap for Semiconductors, 2012. URL ⟨http://www.itrs.net⟩.

[2] OVP, Open Virtual Platforms (OVP), 2013.URL ⟨http://www.ovpworld.org⟩.

[3] D. Diamantopoulos, E. Sotiriou-Xanthopoulos, K. Siozios, G. Economakos, D. Soudris, Plug & Chip: a framework for supporting rapid prototyping of 3d hybrid virtual SoCs, ACM Trans. Embed. Comput. Syst. 13 (5s) (2014) 168:1–168:25. http://dx.doi.org/10.1145/2661634.

[4] F. Slomka, M. Dorfel, R. Munzenberger, R. Hofmann, Hardware/software codesign and rapid prototyping of embedded systems, IEEE Des. Test. Comput. 17 (2) (2000) 28–38. http://dx.doi.org/10.1109/54.844331.

[5] J. Weidendorfer, M. Kowarschik, C. Trinitis, A tool suite for simulation based analysis of memory access behavior, in: Proceedings of the 4th International Conference on Computational Science – ICCS 2004, Krakw, Poland, June 6–9, 2004, Part III, Springer Berlin Heidelberg, Berlin, Heidelberg, 2004, pp. 440–447.

[6] N. Nethercote, J. Seward, Valgrind: A framework for heavyweight dynamic binary instrumentation, in: Proceedings of the 28th ACMSIGPLAN Conference on Programming Language Design and Implementation, PLDI'07, ACM, New York, NY, USA, 2007, pp. 89–100. http://dx.doi.org/10.1145/1250734.1250746.

[7] F. Kordon, Luqi, An introduction to rapid system prototyping, IEEE Trans. Softw. Eng. 28 (9) (2002) 817–821. http://dx.doi.org/10.1109/TSE.2002.1033222.

[8] M. Farshbaf, M.R. Feizi-Derakhshi, Multi-objective Optimization of Graph Partitioning Using Genetic Algorithms, in: Proceedings of the Third International Conference on Advanced Engineering Computing and Applications in Sciences, ADVCOMP '09, 2009, pp. 1–6. http://dx.doi.org/10.1109/ADVCOMP.2009.8.

[9] K.D., A.P., S.A., T.M., A fast and elitist multiobjective genetic algorithm: NSGA-II, IEEE Transactions on Evolutionary Computation, 6 (2), 2002, 182–197.

[10] A. Zhou, B.-Y. Qu, H. Li, S.-Z. Zhao, P.N. Suganthan, Q. Zhang, Multiobjective evolutionary algorithms: a survey of the state of the art, Swarm Evolut. Comput. 1 (1) (2011) 32–49. http://dx.doi.org/10.1016/j.swevo.2011.03.001 (URL ⟨http://www.sciencedirect.com/science/article/pii/S2210650211000058⟩.

[11] G. Palermo, C. Silvano, V. Zaccaria, An efficient design space exploration methodology for on-chip multiprocessors subject to application-specificconstraints, in: Symposium on Application Specific Processors, SASP, 2008, pp. 75–82. http://dx.doi.org/10.1109/SASP.2008.4570789.

[12] G. Mariani, G. Palermo, C. Silvano, V. Zaccaria, Multi-processor system-on-chip DesignSpace Exploration based on multi-level modeling techniques, in: International Symposium on Systems, Architectures, Modeling, and Simulation, SAMOS'09 2009, pp. 118–124. http://dx.doi.org/10.1109/ICSAMOS.2009.5289222.

[13] Xilinx, Inc., URL ⟨http://www.xilinx.com⟩.

[14] V. Chaudhary, J. K. Aggarwal, Parallelism in computer vision: a review, in: V. Kumar, P. S. Gopalakrishnan, L. N. Kanal (Eds.), Parallel Algorithms for MachineIntelligence and Vision, Springer New York, New York, NY, 1990, pp. 271–309. URL ⟨http://dx.doi.org/10.1007/978-1-4612-3390-9_8⟩.

[15] N. Sundaram, Making computer vision computationally efficient (Ph.D. thesis), UC Berkeley: Electrical Engineering & Computer Sciences, 2012.

[16] C. Harris, M. Stephens, A combined corner and edge detector, in: Proceedings of the Fourth Alvey Vision Conference, 1988, pp. 147–151.

[17] H. Bay, A. Ess, T. Tuytelaars, L. van Gool, Speeded-up robust features (SURF), Comput. Vis. Image Underst. 110 (3) (2008) 346–359.

[18] Valgrind Developers.URL ⟨http://valgrind.org/⟩.

[19] A.B. Kahng, J. Lienig, I.L. Markov, J. Hu, VLSI Physical Design: From Graph Partitioning to Timing Closure, 1st Edition, Springer Publishing Company, Incorporated, 2011.

[20] B.W. Kernighan, S. Lin, An efficient heuristic procedure for partitioning graphs, Bell Syst. Tech. J. 49 (2) (1970) 291–307.

[21] C.M. Fiduccia, R.M. Mattheyses, A linear-time heuristic for improving network partitions, in: Proceedings of the IEEE 19th Conference on Design Automation, 1982, pp. 175–181.

[22] C.M. Fonseca, P.J. Fleming, others, Genetic algorithms for multiobjective optimization: formulation, discussion and generalization, in: ICGA, Vol. 93, 1993, pp. 416–423.

[23] J. Teich, Hardware/Software Codesign: The past, the present, and predicting the future, Proceedings of the IEEE 100 Special Centennial Issue , 2012, 1411–1430. http://dx.doi.org/10.1109/JPROC.2011.2182009.

[24] H.-Y. Liu, M. Petracca, L. Carloni, Compositional system-level design exploration with planning of high-level synthesis, in: Design, Automation Test in Europe Conference Exhibition (DATE), 2012, pp. 641–646. http://dx.doi.org/10.1109/DATE.2012.6176550.

[25] Synopsys Virtualizer. URL ⟨http://www.synopsys.com/Prototyping/VirtualPrototyping/Pages/virtualizer.aspx⟩.

[26] HAPS Family of Physical Prototyping Solutions. URL ⟨http://www.synopsys.com/Prototyping/FPGABasedPrototyping/Pages/HAPS.aspx⟩.

[27] Using Hardware Co-Simulation with Vivado System Generator for DSP.URL ⟨https://www.xilinx.com/video/hardware/hardware-co-simulation-vivado-system-generator-for-dsp.html⟩.

[28] Altera, Altera Virtual Target, 2013, ⟨http://www.altera.com/devices/processor/arm/cortex-a9/virtual-target/proc-a9-virtual-target.html⟩.

[29] SCE-MI, Standard Co-Emulation Modeling Interface (SCE-MI) Reference Manual, 2011, ⟨http://www.accellera.org/downloads/standards/sce-mi/SCE_MI_v21-110112-final.pdf.

[30] T.-C. Yeh, Z.-Y. Lin, M.-C. Chiang, A novel technique for making QEMU an instruction set simulator for co-simulation with system C, in: Proceedings of the International MultiConference of Engineers and Computer Scientists 2011, Vol. I, Hong Kong, 2011, pp. 288–291.

[31] E. Sotiriou-Xanthopoulos, S. Xydis, K. Siozios, G. Economakos, D. Soudris, Rapid prototyping and design space exploration methodologies for many-accelerator systems, in: 2015 Proceedings of the 25th International Conference on Field Programmable Logic and Applications (FPL), 2015, pp. 1–2. http://dx.doi.org/10.1109/FPL.2015.7293990.