UNIVERSIDADE FEDERAL DE PELOTAS
Centro de Desenvolvimento Tecnológico
Programa de Pós-Graduação em Computação

Tese

FPGA-Enabled Heterogeneous System Simulation for Early Design Space Exploration

Carlos Michel Betemps

Pelotas, 2021

Carlos Michel Betemps

FPGA-Enabled Heterogeneous System Simulation for Early Design Space Exploration

Tese apresentada ao Programa de Pós-Graduação em Computação do Centro de Desenvolvimento Tecnológico da Universidade Federal de Pelotas, como requisito parcial à obtenção do título de Doutor em Ciência da Computação.

Advisor: Prof. Dr. Bruno Zatt
Coadvisores: Prof. Dr. Daniel Munari Vilchez Palomino
Prof. Dr. Marcelo Schiavon Porto

Pelotas, 2021

Universidade Federal de Pelotas / Sistema de Bibliotecas
Catalogação na Publicação

B562f Betemps, Carlos Michel

FPGA-Enabled heterogeneous system simulation for early design space exploration / Carlos Michel Betemps ; Bruno Zatt, orientador ; Daniel Munari Vilchez Palomino, Marcelo Schiavon Porto, coorientadores. — Pelotas, 2021.

156 f. : il.

Tese (Doutorado) — Programa de Pós-Graduação em Computação, Centro de Desenvolvimento Tecnológico, Universidade Federal de Pelotas, 2021.

1. Heterogeneous system simulator. 2. PGA simulation. 3. Early design space exploration. 4. Dynamic and partial reconfiguration. 5. Partially reconfigurable regions. I. Zatt, Bruno, orient. II. Palomino, Daniel Munari Vilchez, coorient. III. Porto, Marcelo Schiavon, coorient. IV. Título.

CDD : 005

Elaborada por Maria Inez Figueiredo Figas Machado CRB: 10/1612

Carlos Michel Betemps

FPGA-Enabled Heterogeneous System Simulation for Early Design Space Exploration

Tese aprovada, como requisito parcial, para obtenção do grau de Doutor em Ciência da Computação, Programa de Pós-Graduação em Computação, Centro de Desenvolvimento Tecnológico, Universidade Federal de Pelotas.

Data da Defesa: 15 de março de 2021 (March 15, 2021)

Banca Examinadora:

Prof. Dr. Bruno Zatt (orientador)
Doutor em Microeletrônica pela Universidade Federal do Rio Grande do Sul.

Prof. Dr. Daniel Munari Vilchez Palomino (co-orientador)
Doutor em Computação pela Universidade Federal do Rio Grande do Sul.

Prof. Dr. Marcelo Schiavon Porto (co-orientador)
Doutor em Computação pela Universidade Federal do Rio Grande do Sul.

Prof. Dr. André Luís Del Mestre Martins
Doutor em Ciência da Computação pela Pontifícia Universidade Católica do Rio Grande do Sul.

Prof. Dr. Mateus Beck Rutzig
Doutor em Computação pela Universidade Federal do Rio Grande do Sul.

Profª. Drª. Lisane Brisolara de Brisolara
Doutora em Ciência da Computação pela Universidade Federal do Rio Grande do Sul.

Dedico este trabalho . . .

. . . a minha família – **Carla**, **Heitor**, **Davi** e **Miguel** – vocês representam a minha essência

. . . ao meu pai e irmãos – **Carlinhos**, **Daniel** e **Maurício** – vocês são minha origem e referência

. . . a memória de minha mãe – **Maria Irene** – suas lembranças e doces palavras ainda guiam meu ser

. . . a memória de meus avós e padrinhos **Domingos e Tereza** e minha avó **Nair**

AGRADECIMENTOS

Primeiramente, agradeço ao Bruno Zatt pela troca de experiências e orientação durante este trabalho. Sua perspectiva e conhecimento da área foram essenciais para o desenvolvimento desta tese. Da mesma forma, agradeço ao Daniel Palomino e Marcelo Porto pela co-orientação e por trazerem suas diferentes visões para o contexto do trabalho.

Agradeço aos colegas Mateus Melo e Douglas Costa por compartilharem seus conhecimentos e dedicação em atividades relacionadas a este trabalho. Também agradeço aos demais colegas do PPGC, em especial ao Wagner Penny, Ruhan Conceição e Renato Souza pela acolhida e conversas no dia a dia do laboratório.

Gostaria de agradecer a todos os docentes do PPGC/UFPel, em especial aos professores Julio Mattos, Lisane Brisolara, Paulo Ferreira, Gerson Cavalheiro e Mauricio Pilla por compartilharem suas ideias e conhecimentos nas suas disciplinas do PPGC. Agradeço também a UFPel, instituição que me acolheu durante minha graduação e que também possibilitou o ingresso e formação nesta etapa de doutoramento em minha vida como docente universitário.

Agradeço aos colegas docentes do curso de Engenharia de Computação do Campus Bagé da UNIPAMPA pelo apoio na concretização do doutoramento. Em especial, agradeço a Universidade Federal do Pampa - UNIPAMPA pelo apoio institucional na formação de seus docentes.

Agradeço a todos que direta ou indiretamente me apoiaram para que fosse possível desenvolver este trabalho e todas suas atividades inerentes.

Por fim, como um ato de fé, agradeço a Deus pelas oportunidades de aprendizado que se apresentaram em minha vida e por todas as conquistas alcançadas.

. . .
Different eyes see different things
Different hearts
Beat on different strings
. . .
— GEDDY LEE (RUSH)

ABSTRACT

BETEMPS, Carlos Michel. **FPGA-Enabled Heterogeneous System Simulation for Early Design Space Exploration**. Advisor: Bruno Zatt. 2021. 156 f. Thesis (Doctorate in Computer Science) – Technology Development Center, Federal University of Pelotas, Pelotas, 2021.

Heterogeneous systems architectures usually include processing elements such as Central Processing Units and General Purpose Graphics Processing Units, frequently enabling optimization opportunities in terms of execution time, consumed energy, resource utilization, and throughput. In turn, the heterogeneity brings a series of new design challenges when compared to homogeneous systems. An even more challenging scenario appears when such heterogeneous systems feature hardware acceleration through dynamic and partial FPGA (Field Programmable Gate Array) reconfiguration. This work presents a Modeling and Simulation infrastructure for early Design Space Exploration (DSE) of heterogeneous systems by comprising a methodology to create high-level models of the system and a simulator complying with those models. A designer can partition an FPGA into Partially Reconfigurable Regions (PRRs) that can pass through a Dynamic and Partial Reconfiguration (DPR) during runtime. Considering those aspects, the modeling methodology contains the flow and automatic hardware generation to annotate our simulation models. FEHetSS is an FPGA-Enabled Heterogeneous System Simulator aiming to provide support for decision making in early design phases. We describe FEHetSS presenting its structure, models, and simulation flow. FEHetSS considers the tasks' latencies even those related to reconfiguration, also estimating the processing elements' power and resource utilization. Based on case studies, we demonstrate the methodology and FEHetSS's potentialities. First, we model heterogeneous systems and use FEHetSS as a tool to evaluate single-points in early DSE. Second, we conceive distinct hardware design (e.g., pipelining and parallelism) models for an application kernel and utilize FEHetSS as a tool to assist designers considering a holistic system perspective. Third, we restrict a couple of design spaces applying FEHetSS to perform exhaustive exploration. Last, we prepare a DSE environment integrating an optimization heuristic and FEHetSS, performing simulations based on exploration parameters. Case studies' results and analysis demonstrate the infrastructure features in the modeling and simulation of FPGA-enabled heterogeneous systems.

Keywords: Heterogeneous System Simulator. FPGA Simulation. Early Design Space Exploration. Dynamic and Partial Reconfiguration. Partially Reconfigurable Regions.

RESUMO

BETEMPS, Carlos Michel. **Simulação de Sistemas Heterogêneos Habilitados para FPGA visando Exploração Precoce do Espaço de Projeto** . Orientador: Bruno Zatt. 2021. 156 f. Tese (Doutorado em Ciência da Computação) – Centro de Desenvolvimento Tecnológico, Universidade Federal de Pelotas, Pelotas, 2021.

Arquiteturas de Sistemas Heterogêneos usualmente incluem CPUs e GPUs como elementos de processamento habilitando oportunidades de otimização no tempo de execução, energia consumida, utilização de recursos e desempenho. Em comparação ao projeto de sistemas homogêneos, novos desafios surgem com a heterogeneidade, sendo estes potencializados pela inclusão de aceleradores de hardware como FPGAs. Este trabalho apresenta uma infraestrutura de Modelagem e Simulação em alto nível para a Exploração Precoce do Espaço de Projeto de sistemas heterogêneos habilitados para incluir dispositivos FPGA. Tais dispositivos podem ser particionados em Regiões Parcialmente Reconfiguráveis (PRR) que, por sua vez, podem passar por Reconfiguração Dinâmica e Parcial (DPR). Considerando esses aspectos, a metodologia inclui o fluxo de modelagem e a geração automática de hardware necessários às anotações dos modelos de simulação. FEHetSS é um simulador de sistemas heterogêneos habilitado para FPGAs que fornece suporte à tomada de decisão em estágios iniciais de projeto, sendo este detalhado em relação a sua estrutura, modelos e fluxo de simulação. Nas simulações são consideradas as latências das tarefas de aplicação e aquelas referentes às reconfigurações, assim como o consumo energético e utilização de recursos do sistema. Estudos de caso demonstram as capacidades da infraestrutura. Primeiro, foram criados modelos de sistemas heterogêneos para simulação via FEHetSS atuando como uma ferramenta de avaliação na exploração precoce do espaço de projeto. Segundo, foram concebidos diferentes modelos de hardware para o kernel de uma aplicação exemplo, sendo estes submetidos ao FEHetSS para a avaliação do projeto em uma perspectiva holística do sistema. Terceiro, foram definidos parâmetros de busca restringindo o espaço de projeto para exploração exaustiva com FEHetSS. Por último, FEHetSS foi integrado em um ambiente gerenciado por heurística de otimização visando a exploração baseada em parâmetros do espaço de projeto. Os resultados e respectivas análises demonstraram as potencialidades da infraestrutura na modelagem e simulação de sistemas heterogêneos habilitados para FPGAs dinamicamente reconfiguráveis e particionados em PRRs.

Palavras-chave: Simulador de Sistemas Heterogêneos. Simulação FPGA. Antecipada Exploração do Espaço de Projeto. Reconfiguração Dinâmica e Parcial. Regiões Parcialmente Reconfiguráveis.

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS AND ACRONYMS

ALUT	Adaptive Look-Up Tables
aoc	Altera Offline Compiler
ASIC	Application Specific Integrated Circuits
BIHW	Board Interface Hardware
CPU	Central Processing Unit
DPC	Dynamic Power for the Core
DPPH	Dynamic Power for the Periphery Hardware
DPR	Dynamic and Partial Reconfiguration
DSE	Design Space Exploration
DSP	Digital Signal Processor
ES	Embedded Systems
FEHetSS	FPGA-Enabled Heterogeneous System Simulator
FPGA	Field Programmable Gate Array
GPGPU	General Purpose Graphics Processing Unit
GPU	Graphics Processing Unit
HW	Hardware
HTLP	Heterogeneous Task-Level Parallelism
I/O	Input/Output
LLVM	Low-Level Virtual Machine
LUT	Look-Up Tables
MPSoC	Multiprocessor System-on-Chip
OS	Operating System
OpenCL	Open Computing Language
PF	Pareto Front
PE	Processing Element
PR	Partial Reconfiguration

PRM	Partially Reconfigurable Module
PRR	Partially Reconfigurable Region
RTL	Register-transfer Level
SA	Simulated Annealing
SIS	System-level Simulation
SoC	System-on-Chip
SPD	Standby Power for the Device
SPPH	Standby Power for the Periphery Hardware
SW	Software
TLM	Transaction-Level Modeling
TLP	Task-Level Parallelism
VirtualHWAccel	Virtual Hardware Accelerator
VP	Virtual Platform
XML	eXtensible Markup Language

CONTENTS

1 INTRODUCTION

Today's computing environments explore the capabilities of a wide range of processors – such as CPUs (Central Processing Units), DSPs (Digital Signal Processors), media processors, vector processors, reconfigurable hardware (such as FPGAs - Field Programmable Gate Arrays), and General Purpose Graphics Processing Units (GPGPUs, or just GPUs) (KAELI et al., 2015; WAIDYASOORIYA; HARIYAMA; UCHIYAMA, 2018). These systems integrate a relevant number of independent and heterogeneous processing resources within the same environment (MIELE et al., 2015).

The workloads submitted to the systems usually present a series of different behaviors. It comprises applications with a mix of computational demands such as control-intensive (e.g., searching, sorting, and parsing), data-intensive (e.g., image processing, simulation and modeling, and data mining), and compute-intensive tasks (e.g., iterative methods, numerical methods, and financial modeling) (KAELI et al., 2015). Thus, different workloads characterize the computing scenarios – consisting of a range of applications with distinct performance requirements and varying amounts of data to process (BOLCHINI et al., 2015), demanding diverse processing capabilities. In these scenarios, the overall throughput is subject to the hardware (HW) devices' computational efficiency. Moreover, there is no single architecture suitable for all types of applications (KAELI et al., 2015).

Some applications, especially those that contain substantial fractions of sequential code, run faster on the CPU. Others have extensive use of data parallelism finishing their tasks earlier by running on Graphical Processing Units (GPUs) (SINGH et al., 2017). Besides, several parts (excerpts) of the applications can be executed in parallel – the so-called *kernels* of the applications. It allows the exploitation of task-level parallelism (TLP) (SCARPAZZA et al., 2006). The multi-processed systems, including those with a series of heterogeneous processing elements (PEs), exploit TLP.

Typically, heterogeneous systems better meet the workload's demands. Computing with heterogeneous resources such as FPGAs and GPUs can improve system performance and energy efficiency (DURELLI et al., 2014; WAIDYASOORIYA; HARIYAMA; UCHIYAMA, 2018). Providing a high computing capacity and energy efficiency, usually

enabled by abundant parallelism of data (ROGERS, 2013), processing elements (PEs) such as GPUs and FPGAs arise in the heterogeneous context with a very motivational perspective. Thus, a promising approach to deal with current computing scenarios is to exploit specialized computing resources integrated into heterogeneous system architectures, such as asymmetric multi-core CPUs, specialized graphics coprocessors, GPUs, and reconfigurable hardware like FPGAs (BOLCHINI et al., 2015). Multiprocessor System-on-Chip (MPSoC) is an instance of heterogeneous systems providing large amounts of computing capability (hsafoundation, 2020). These systems commonly contain CPUs (Central Processing Units) cores of different characteristics, GPGPUs (General Purpose Graphics Processing Units) or GPUs (Graphics Processing Units) processors, and perhaps other PEs – e.g., FPGAs (Field Programmable Gate Arrays) devices, ASIC (Application Specific Integrated Circuit) HW accelerators, and DSPs – Digital Signal Processors.

FPGA devices can provide enhanced performance combined with superior power efficiency (MUSLIM et al., 2017), whereas keeping flexibility (not possible to ASIC accelerators). These aspects plausibly have led to the deployment of FPGAs in cloud computing architectures like F1 Instances of Amazon (AWS) (AMAZON, 2020), and Project Catapult of Microsoft Research (MICROSOFT, 2020). Nonetheless, programming FPGAs usually requires a low-level HW description language (e.g., Verilog or VHDL) (SHATA; ELTEIR; EL-ZOGHABI, 2019). The design initiatives using this abstraction level demands longer development time. However, current High-level Synthesis (HLS) tools have enabled the programming of these devices. It allows the use of a high-level abstraction language in FPGA development. Indeed, FPGA vendors currently release mature environments (INTEL, 2020a) (XILINX, 2020) that allow HW description and synthesis in higher-level languages, such as OpenCL, C, and C++.

The design of a new HW/SW (hardware/software) system, such as an embedded system, is a complex initiative. The exploring of alternative solutions to such systems frequently occurs at a system-level abstraction (GRIES, 2004). System-level Simulation (SIS) uses coarser models for architecture and applications (GRIES, 2004), allowing reduced time of simulation and possibly a broader evaluation of the design alternatives. The referred models require non-functional requirements that may include latency, power estimates, HW area, and possibly others. Also, it depends on the system workload and the target platform. We can obtain estimations data using platform execution, cycle-accurate simulation, tool estimation, or even extracting from component datasheets and academic works. Moreover, the employed modeling paradigm must include a framework containing concepts and respective representations, providing the fundamental elements to describe the platform in its parts, such as architecture and workload.

Design Space Exploration (DSE) is the process of finding a set of optimal (or near-

optimal) configurations (the *Pareto* front) regarding some evaluation metrics and following design restrictions (SINAEI; FATEMI, 2018). DSE must evaluate the solutions based on multiple optimization objectives – such as execution time, energy consumption, utilization, cost, and others. Frequently, the objectives conflict and there is no single solution that optimizes all (PIMENTEL, 2017). Resuming, we can state that the DSE goal is to find the *Pareto* front (or the most proximal settings) of the design space (PIMENTEL, 2017). Several works use heuristics to accelerate the DSE process such as Simulate Annealing (SA) (SINAEI; FATEMI, 2018), Genetic Algorithms (GA) (JIANG; ELES; PENG, 2016), and Ant Colony Optimization (ACO) (WANG et al., 2007). These heuristics provide a search mechanism that systematically traverses the design space aiming to find quality solutions.

Cycle and instruction accurate simulators frequently cannot deal with the demand under DSE, given its long simulation time, architecture details (e.g., aspects of memory, cache configurations, communication channels, etc.), low-level programming/parameterization, and so on. Initial design phases do not allow the time to define all these system details. Many works present simulators and frameworks aiming at the evaluation of architecture settings.

Some works employ low-level simulation, usually at register-transfer level (RTL), to evaluate the architecture settings – such as FENG et al. (2017), LIANG et al. (2017), SHAO et al. (2016), MAKNI et al. (2018), ZHONG et al. (2016), ZHAO et al. (2020), HUANG et al. (2019), LIGNATI et al. (2021), QUADRI et al. (2010), GONG; DIESSEL (2014), and BRITO et al. (2007a,b)). Therefore, they likely demand long simulation times and more detailed system models/artifacts, caused by the simulated details. In contrast, works like MIELE et al. (2015), SIGDEL et al. (2009a,b), DUHEM et al. (2015), NOGUEIRA et al. (2016), AN; GAMATIÉ; RUTTEN (2015), and BLOUIN et al. (2011) employ high-level simulation also using models of high-abstraction level. Such approaches usually require less efforts in the models' description and the simulation itself. Others works utilize standard languages as UML (Unified Modeling Language) (HUANG; HSIUNG; SHEN, 2010) and their extension UML/MARTE (Modeling and Analysis of Real-Time and Embedded systems) (GRÜTTNER et al., 2013; HERRERA et al., 2014) (QUADRI et al., 2010) to model the platform using a series of model transformations to generate simulation models, but most employing low-level abstractions.

Several works do not account for the system power/energy. But, those works that do it usually employ some kind of component characterization (MIELE et al., 2015; SHAO et al., 2016; NOGUEIRA et al., 2016; AN; GAMATIÉ; RUTTEN, 2015; GRÜTTNER et al., 2013; MAKNI et al., 2018; LIGNATI et al., 2021) or even direct interaction with HW (HUANG; HSIUNG; SHEN, 2010) to generate power/energy-related metrics.

FPGAs are configurable components. It can provide higher flexibility degree when

Dynamic and Partial Reconfiguration (DPR) and Partially Reconfigurable Regions (PRRs) are in use. DPR allows the configuration of a device's portion while still operating in other regions. PRRs are portions of the device that can pass by a DPR. Although the previous works deal with FPGA fabric, DPR and PRR features are not always present. SIGDEL et al. (2009a) and DUHEM et al. (2015) deal with both features but lack in power estimation. AN; GAMATIÉ; RUTTEN (2015), BLOUIN et al. (2011), LIANG et al. (2017), FENG et al. (2017), and QUADRI et al. (2010) consider DPR feature, but with gaps in the use of PRRs or even the accounting of reconfiguration time/energy. GONG; DIESSEL (2014) and BRITO et al. (2007a,b) only focus on DPR issues, not providing a wide system-level view. The goal in works like MAKNI et al. (2018), ZHONG et al. (2016), and (ZHAO et al., 2020) is the FPGA's Performance, Power, and Area estimations, while in LIGNATI et al. (2021) the focus is on resource provisioning in CPU-FPGA-based Cloud Systems.

Thus, these works lack some essential aspects during the evaluation of Heterogeneous System Architectures (HSA) in early design phases, as follows: the necessity of low-level designing (e.g., to describe a task using HDL – hardware description languages, or describing floorplanning aspects), time-consuming evaluation due to detailed simulations, absence of power modeling, the impossibility of deal with multiple Partially Reconfigurable Regions (PRRs) or to model Dynamic and Partial Reconfiguration (DPR), and even the non-compliance with HW (e.g., FPGA) accelerated systems.

FPGA-enabled system simulation appears to be a promising feature due to the possibility of using Dynamic and Partial Reconfiguration (DPR), or even the ability to define one or more Partially Reconfigurable Regions (PRRs). Thus, a flow and a simulator proficient in modeling an FPGA-capable system using a high abstraction level and simulating its behavior in terms of latency, power consumption, and HW utilization are of paramount importance. Regarding the embedded systems domain, the employment of large FPGA devices is usually not an option, hence, hampering the acceleration of whole applications. Thus, wondering about a modeling methodology, the OpenCL (KHRONOS, 2020) models and concepts appear to be suitable in this context where a designer shall models an FPGA unit accelerating (only) the application's kernels as HW tasks, maintaining the remaining running in a general-purpose processor.

1.1 Thesis' Research Challenges

We present the Research Challenges of the Thesis in this section, passing before by a Motivational Scenario that contextualizes the work.

1.1.1 Motivational Scenario

Consider a design initiative for a new embedded system. It must deal with a specific workload and considers metrics such as execution time, power/energy consumption, and processing elements/HW area utilization. Since probably no physical architecture is available, early design evaluations must happen using high-level models. Thus, System-Level Simulation is a way to assess the design points employing Virtual Platforms (VP) modeling the system architecture, the workload, and their mappings. With notions such as host/device processors and application kernels, OpenCL concepts serve as a basis for modeling the system architecture&workload. Also, considering High-Level Synthesis tools, it outcomes appropriate artifacts that enable modeling the application's kernel(s) running in FPGA units. FPGAs are customizable devices usually providing performance and power efficiency. Besides, it offers features like Partially Reconfigurable Regions (PRRs) and Dynamic and Partial Reconfiguration (DPR). Each PRR can potentially act as an independent HW accelerator. DPR enables dynamically change the PRR's configuration. Moreover, the custom capability of FPGA devices allows experimenting with diverse HW task implementations aiming to obtain a good trade-off between execution time, energy consumption, and resource utilization. Given this scenario, how could a designer conduct early design space exploration (early-DSE) activities in a feasible time considering high-abstraction descriptions of the system and its applications, producing valuable design artifacts for initial decisions making? All of this, before passing to more detailed and low-level HW/SW development.

1.1.2 Research Challenges

We define the Thesis Research Challenges through the following items:

- **Challenge 01 (C01)**: *How to enable early-DSE on heterogeneous systems that include reconfigurable hardware acceleration through FPGAs?*
- **Challenge 02 (C02)**: *How to assess (at early stages) different architectural FPGA implementations for a specific application kernel, as an HW task, within a heterogeneous system?*
- **Challenge 03 (C03)**: *How to model FPGA units at a high level, supporting PRRs and DPR, focusing on high-level system simulation?*

1.2 Work's Main Contributions

In this work, we describe a methodology to construct models for *Architecture* and *Workload* that can be employed in System-Level Simulation through FEHetSS (FPGA-Enabled Heterogeneous System Simulator) simulator. In its simulations, FEHetSS

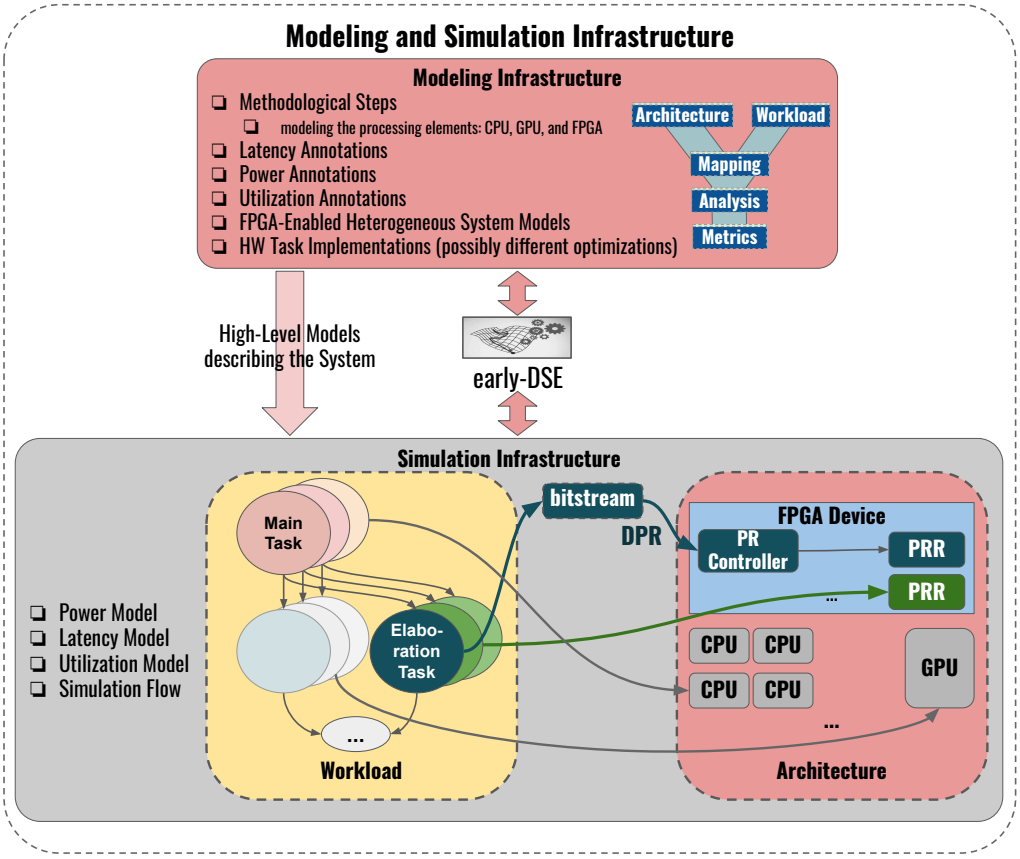


Figure 1 – Thesis Contributions Context

may include, beyond CPUs and GPUs, virtual hardware accelerators (*VirtualHWAccels*) modeled as FPGA devices. Our methodology includes the high-level modeling of FPGA units featuring PRRs and DPR, also supported by FEHetSS. As part of case studies, we describe the main capabilities of FEHetSS. Towards this end, our work makes some contributions which are illustrated in Fig. 1, as a context diagram, and listed in the following items:

- A Modeling and Simulation Infrastructure to support early-DSE of FPGA-capable Heterogeneous Systems;
- A set of methodological steps aiming the high-level modeling of the system *Architecture* and the *Workload* applications;
- The task-level modeling of an application considering sequential management tasks (called main tasks) and heterogeneously accelerated potentially paralleled tasks (named elaboration tasks);
- The high-level modeling of FPGA units supporting Dynamic and Partial Reconfiguration (DPR) and the device partition in Partially Reconfigurable Regions (PRRs);

- The FEHetSS simulator and its simulation model for FPGA-Enabled Systems, considering aspects such as latency, power, and resource utilization;
- A set of Case Studies demonstrating the modeling and simulation capabilities of the proposed infrastructure (FEHetSS and models' creation methodology), as follows:
  - (*i*) DSE of heterogeneous settings using FEHetSS as a single design point assessing tool;
  - (*ii*) Evaluation of HW Design Implementations of an application's kernel, supported by the modeling steps and our simulator;
  - (*iii*) Exhaustive Search of solutions in a customized design space employing the rapid simulations of FEHetSS;
  - (*iv*) Embedding of FEHetSS in a Heuristic-based DSE Environment performing explorations in a parametrized design space.

In summary, *the main goal of this work is to provide a modeling and simulation infrastructure to perform early Design Space Exploration (DSE) for heterogeneous systems featuring FPGA fabric for reconfigurable hardware acceleration. Moreover, the framework also enables the design's implementation assessment by considering the hardware/software interface in a heterogeneous architecture.*

1.3 Work's Methodology Overview

For the development of the Thesis, we employ a set of tools, processes, and artifacts aiming to attend the presented Research Challenges. Since our main goal is to explore design spaces of FPGA-enabled heterogeneous systems by simulation in the early phases of development, we base our modeling steps on concepts that are commonly employed in programming languages like OpenCL (KHRONOS, 2020) - e.g., applications with portions of *host code* to manage and prepare the execution of the *kernel code* which is possibly *accelerated* in *devices* like CPU, GPU, or FPGAs. To make it possible to annotate our models, especially those related to FPGA devices, we employ High-Level Synthesis (HLS) (INTEL, 2020a) tools to make estimations for the models, providing details such as latency, power, and HW utilization. Besides, regarding CPU and GPU architectural elements, we use a real platform (HARDKERNEL, 2020) to extract measurements for the characterization of the processors' power and the applications' tasks latencies.

We employ the Y-Chart approach (GRIES, 2004) to model our platforms, using models to separately describe the architectural elements – such as the Processing Elements (PEs) – and the workload – a set of applications. Our modeling elements

include the characterization of CPU, GPU, and FPGA, describing them through power models containing the operation power, idle power, and frequency. For the description of FPGA devices in the models, extra annotations depict aspects such as: (*i*) the necessary HW resources to implement a custom processor within the device, (*ii*) the annotations to describe the employment of Dynamic and Partial Reconfiguration (DPR), (*iii*) the device partition in Partially Reconfigurable Regions (PRR), (*iv*) the size of the called bitstream files used to (re)configure the whole device or its PRRs, and others.

To model the applications, we employ task graphs where each task represents an application's portion at a coarse-grained level – e.g., a whole procedure. A performance model describes each task pointing out a PE type and model, as well as the operating frequency and respective latency. Specialized tasks, called elaboration tasks, model the application kernels allowing the mapping on distinct PEs (e.g., CPU, GPU, and FPGA).

On top of our previously developed SAVE-htlp (BETEMPS et al., 2018) framework, we develop the FEHetSS simulator using SystemC at Transaction Level Modeling (TLM) abstraction level. Aiming to promote an agile utilization of FEHetSS, we prepare a repository containing Architecture and Workload model portions capable to be reused in the generation of different solutions to be simulated with FEHetSS. A set of automation scrips accompany our model repository allowing us to automate the interaction with FEHetSS.

As a proof of concept regarding the modeling methodology and aiming to experiment simulations with FEHetSS, we prepare a set of Case Studies with FEHetSS also employing the generation of solutions from our model repository. Using the GQM (Goal-Question-Metric) approach (BASILI; CALDIERA; ROMBACH, 1994), we define the `Goal`(s) of each Case Study by setting up `Question`(s) to be answered and the `Metric`(s) to be extracted during the case studies. The answers to the Case Studies Questions give us support for the fulfillment assessment of the Thesis Research Challenges.

The repository automation facilitates the performing of the Case Studies, especially in the handle of a design space exhaustive traversing using FEHetSS, as well as an integration of FEHetSS in a heuristic-based DSE environment. We employ Simulated Annealing (SA) (FRANZIN; STüTZLE, 2019) as the DSE environment's heuristic. SA is a simple to formulate, robust, and compact strategy for single or multiple objective optimization problems. It is a widely used heuristic commonly providing good solutions even for mixed problems of discrete and continuous spaces (SUMAN; KUMAR, 2006; FRANZIN; STüTZLE, 2019).

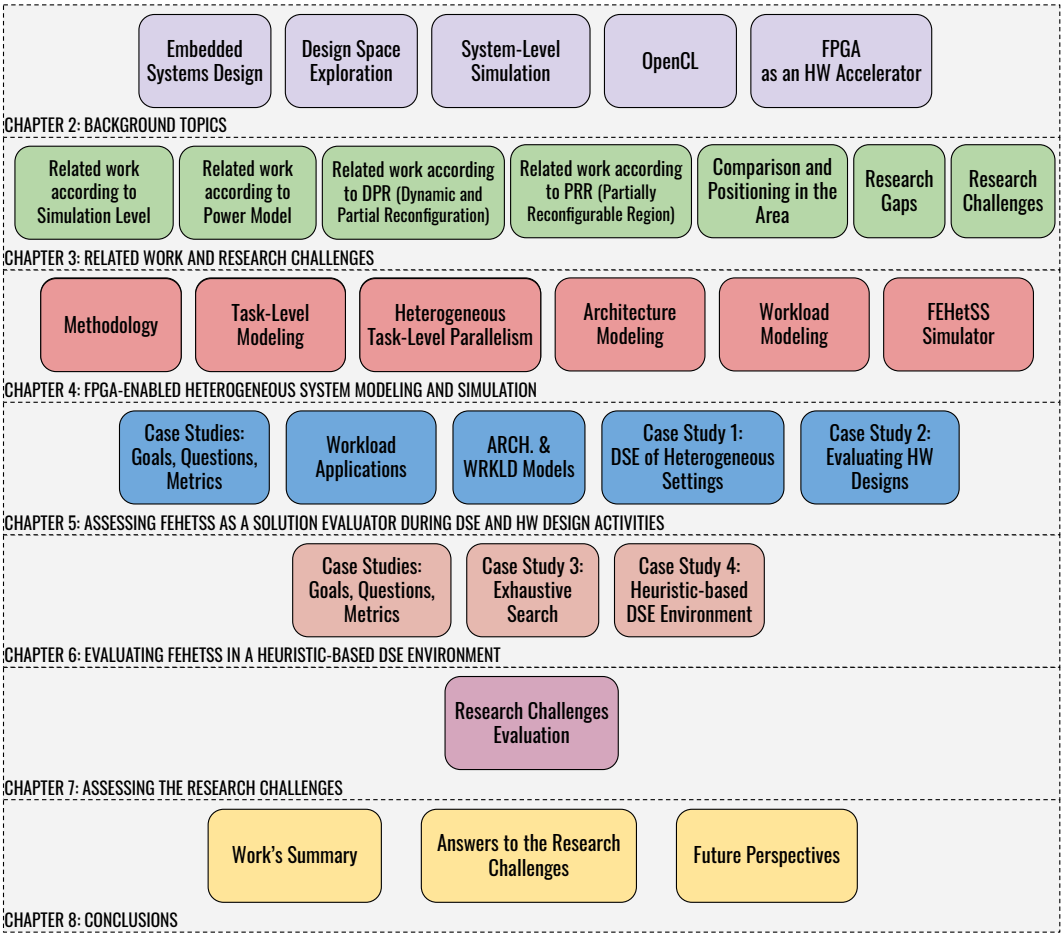


Figure 2 – Thesis' Structure and Chapters' Main Topics

1.4 Thesis Structure

The Thesis is structured as follows. Chapter 2 presents background topics of the work. Next, in Chapter 3, we describe the related works and present the Thesis' Research Challenges. Chapter 4 presents the work's main contributions passing by sections approaching the Thesis' methodological view, the modeling of heterogeneous task-level parallelism, the methodology to model Virtual Platforms, and the FEHetSS simulator. Chapter 5 describes the experimentation of FEHetSS by manual DSE in Case Studies #1 and #2, also presenting its results. After, Chapter 6 presents the Case Studies #3 and #4 that automatically employ FEHetSS through an exhaustive search and within a heuristic-based DSE environment, respectively. In Chapter 7, we present an assessment of the Thesis' Research Challenges based on the Case Studies results and respective answered questions. Chapter 8 concludes the work by presenting answers for the Research Challenges, also indicating some future perspectives for the Thesis. Fig. 2 highlights the main topics approached in each chapter.

2 BACKGROUND TOPICS

This chapter presents some background aspects related to the thesis. Embedded Systems (ES) are composed of SW and HW components, possibly including heterogeneous Processing Elements (PEs). The design of this type of system (Sec. 2.1) involves the analysis of the feasible design points (solutions). Usually, the design space of these systems is vast, with multiple objective functions to optimize. The Design Space Exploration (DSE) (Sec. 2.2) is the process of evaluating the design points and defining which one is the best trade-off. During the DSE, heuristics (Sec. 2.2.1) can prune the design space and still provide quality solutions. Besides, System-level Simulation (Sec. 2.3) can rapidly assess the design points based on high-level models describing Virtual Platforms (VPs) that include models for the architecture, the workload, and the mappings between both. The OpenCL standard (Sec. 2.4) provides a programming model suitable to employ in the modeling of workload's applications – describing tasks that manage the execution (main tasks) and the ones that represent the application's kernel (elaboration tasks). Elaboration tasks can run in different PEs, allowing exploring the system's heterogeneity. Besides, modern OpenCL programming environments allow employing application kernels as HW tasks using the FPGA devices flexibility (Sec. 2.5), making it act like an HW Accelerator. Also, Dynamic and Partial Reconfiguration (DPR) enhances such flexibility even more. DPR allows configuring multiple regions (called Partially Reconfigurable Regions - PRRs) separately and dynamically. Indeed, each PRR can implement an HW task separately, making a single FPGA device acting as several HW accelerators.

2.1 Embedded Systems and its Design

Embedded Systems (ES) provide specific and dedicated functions inside a host (CAMPOSANO; WILBERG, 1996), being a component of a greater system controlling and monitoring it (GUESSI et al., 2012). ES are information processing systems enclosed in a larger product, usually not visible to its users (MARWEDEL, 2006).

ES are highly diverse, varying from little systems – such as refrigerator controllers,

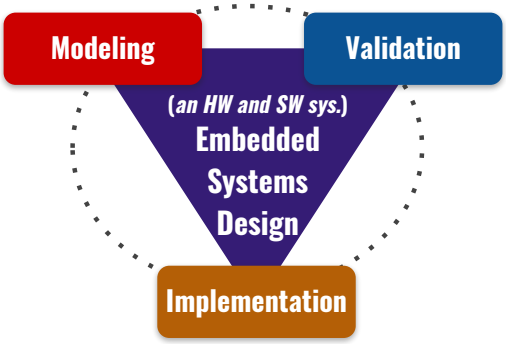


Figure 3 – Embedded Systems Design.

ATMs, or intelligent sensors in construction automation – to complex and distributed systems – e.g. automotive control and aerospace systems (VOELTER et al., 2013). Embedded software is constantly growing in terms of functionality, size, and complexity (HENDRIKS et al., 2016). Thus, ES design is in a competitive context that rapidly evolves as new technologies are introduced (BERTELS, 2012). The transformation of a design initiative in a successful product depends on the time-to-market as new functionalities are proportioned obeying performance restrictions and accessible prices (HERRERA et al., 2014; MISCHKALLA; HE; MUELLER, 2010).

According to Fig. 3, as systems containing HW and SW components, the design of Embedded Systems involves modeling, validation, and implementation (DE MICHELI; GUPTA, 1997). **Modeling** deals with the conceptualization and refinement of the specification, producing models of HW and SW. **Validation** aims to reach a reasonable level of reliability that the system will run as designed. **Implementation** is the system's physical realization in terms of hardware and software. This thesis involves aspects of modeling and validation regarding ES development.

Due to the complexity of the design of new products, the competitiveness in the ES market, and the gap in the joint design of hardware and software, more and more tasks of exploring solutions are carried out at the system level (GRIES, 2004; GAJSKI et al., 2009; SINAEI; FATEMI, 2018), to enable decision making in the early stages of development. In this context, models with a high level of abstraction are used to describe the different architecture configurations and workloads to be submitted to the system. As evaluations need to occur in the early stages of development, when usually there is no availability of architecture and applications that can be executed, the use of system-level simulation appears as an alternative for evaluating solutions during Design Space Exploration (DSE).

2.2 Design Space Exploration

Design Space Exploration (DSE) is the process to find a set of optimal (or near-optimal) configurations (the *Pareto* front) regarding some evaluation metrics and following design restrictions (SINAEI; FATEMI, 2018)(PANERATI; SCIUTO; BELTRAME, 2017). It is the process of analyzing the set of possible solutions and defining which one will be selected (MARWEDEL, 2006). The solutions in the *Pareto* front are the optimal points that are not dominated by any other in the design space (PANERATI; SCIUTO; BELTRAME, 2017). A set of metrics (or objective functions) express the quality of a solution. Thus, DSE evaluates the solutions based on multiple optimization objectives – such as execution time, energy consumption, utilization, cost, and others. It characterizes an optimization problem (PALERMO; SILVANO; ZACCARIA, 2009) whose aim is maximization or minimization. Frequently the objectives conflict, and there is no single solution that optimizes all (PIMENTEL, 2017) – potentially exist several optimal solutions.

In a Multiobjective Optimization Problem (MOP), there is a solution space X in which solutions have m decision variables (or independent variables). Moreover, the DSE searches for a solution or solutions that minimizes (or maximizes) the n objective values (or dependent variables), in the objective space Y, by evaluating objective functions f_i with $1 \leq i \leq n$. Thus, during a DSE we search for MOP solutions characterized by (PIMENTEL, 2017):

$$\begin{aligned} Minimize\ y = f(x) = (f_1(x), f_2(x), \ldots, f_n(x)) \\ where\ x = (x_1, x_2, \ldots, x_m) \in X \\ y = (y_1, y_2, \ldots, y_n) \in Y \end{aligned}$$

Aiming to evaluate the solutions during the DSE, the Pareto dominance relation is usually used. A solution $x_1 \in X$ is said to dominate solution $x_2 \in X$ if and only if $x_1 < x_2$ (PIMENTEL, 2017).

$$\begin{aligned} x_1 < x_2 \Leftrightarrow \forall i \in \{1, 2, \ldots, n\} : f_i(x_1) \leq f_i(x_2) \wedge \\ \exists i \in \{1, 2, \ldots, n\} : f_i(x_1) < f_i(x_2) \end{aligned}$$

Figure 4 illustrates the Pareto front and dominance relation concepts regarding a two objective (f_1 and f_2) problem. Solutions A to F are in the Pareto front, being not comparable with each other. These solutions cannot be ordered since each one is better than the others in at least one objective function but worst in the other one(s). Solutions B to D dominates H, which dominates solutions M to O. Solutions in the light

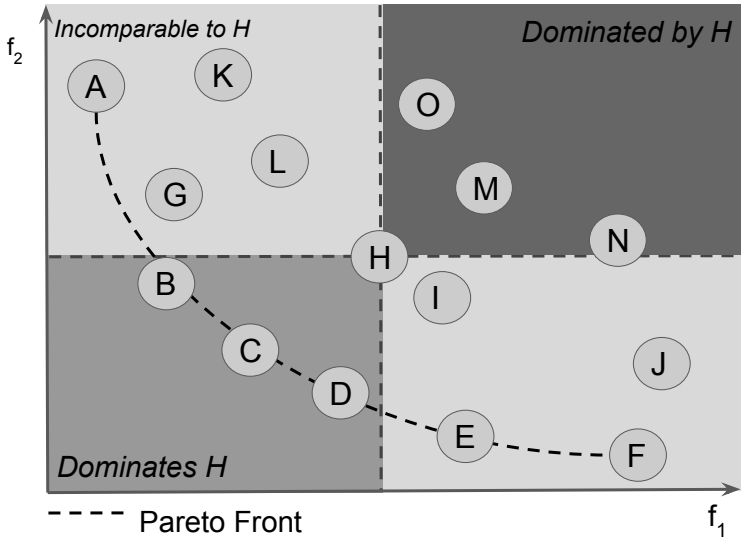


Figure 4 – Pareto Front and Domination relation considering the objective functions f1 and f2, excerpted from (PIMENTEL, 2017).

grey areas are incomparable to H.

The performance of a system is dependent on a diverse set of factors, such as the application architecture, platform HW architecture, which processing elements (e.g. CPU, GPU, FPGA, ASIC, and DSP) are employed for each application task, beyond parameters such as caches and memory sizes. Thus, DSE is a key activity to evaluate the system under development since early design phases (HERRERA et al., 2014; SINAEI; FATEMI, 2018). The DSE adjusts the system parameters aiming at the optimal setting. In a general way, a DSE approach consists of four components (ASCIA et al., 2011): *i*) an entry point represented by the initial configurations, (*ii*) an evaluation model for the settings, (*iii*) an exploration strategy aiming to visit the design space via transformation in the configurations, and (*iv*) a stopping criterion.

According to Fig. 5, DSE usually follows the Y-chart approach (GRIES, 2004). Y-chart separates the architecture and workload specifications, considers the mapping between both, and succeeds in a performance analysis generating performance numbers that can guide changes in the specifications or the mappings. An application model(s) represents the system *workload* – derived from the target application(s) – describing the functional behaviour independently from the architecture. Also, an *architecture* model defines the system resources in the architecture (the processing elements – PEs) and captures its performance constraints (JIA et al., 2013). The *mapping* model links the previous ones allocating tasks in the architecture PEs (GRIES, 2004) for execution (e.g. through simulation). The execution generates metrics for quantitative assessment (JIA et al., 2013). Based on execution metrics, e.g., performance, power, energy, and utilization – obtained by real hardware execution, simulation, or estimates – an analysis is performed allowing to tuning the models for the architecture,

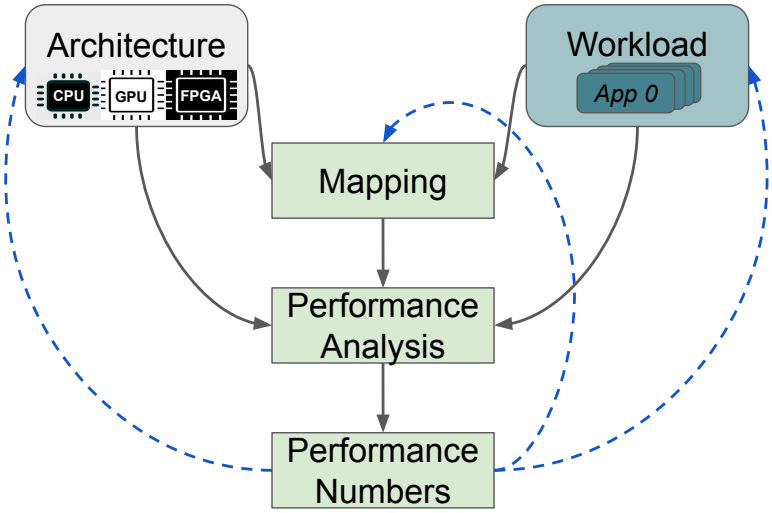


Figure 5 – Y-Chart Approach (based on GRIES (2004))

workload, or mapping.

Obtaining the optimal *Pareto curve* requires the exhaustive search of it within the design space (PIMENTEL, 2017) of a given system. However, the vast size of the design space is a problem in most embedded system designs, and it can become even more severe according to the non-functional design requirements (restrictions) (GLAß et al., 2017; SINAEI; FATEMI, 2018). Thus, the main objective of the DSE can be relaxed in the sense of seeking to minimize the exploration time while guaranteeing good quality solutions (ASCIA et al., 2011) (not necessarily optimal). In general, DSE approaches use strategies to prune the design space and minimize the number of configurations to evaluate (ASCIA et al., 2011; SINAEI; FATEMI, 2018). In this context, we can apply Heuristics (and Meta-Heuristics). Meta-heuristics are general, higher-level methodologies that can guide strategies in the development of heuristics to solve specific optimization problems (TALBI, 2009).

Therefore, the DSE process consists of two interdependent components that need to be accelerated or improved, aiming to deal with the vast design space and to find optimal (or close to it) solutions (GRIES, 2004; SINAEI; FATEMI, 2018):

- (*i*) the evaluation of each design point (configuration) of the solution space using, for example, analytical models or simulation; and
- (*ii*) the search engine that systematically traverses the design space to find quality solutions.

Considering the component (*i*), the use of system-level simulation has interesting characteristics concerning aspects such as simulation time and the use of high-level abstraction models (made with less effort in the early stages of development). As for the component (*ii*), the use of heuristics for selecting and directing searches for

candidate solutions can prune the design space (usually vast in the case of ES) to be visited (examined).

2.2.1 A DSE Heuristic: Simulated Annealing

Meta-heuristics are solution methods that control the interaction between local improvement procedures and high-level strategies to create processes capable of escaping from local optimal and that perform a robust search for a solution space (GENDREAU; POTVIN, 2010, p. vii). On the other hand, TALBI (2009) defines metaheuristics as general high-level methodologies (templates) used as guiding strategies in the development of related heuristics to solve specific optimization problems.

A widely used metaheuristic is **Simulated Annealing** (SA), which was adapted to solve combinatorial optimization problems along the years (FRANZIN; STüTZLE, 2019). SA is a compact and robust technique for obtaining excellent solution(s) for single or multiple objective optimization problems (SUMAN; KUMAR, 2006). The technique is based on an analogy of thermodynamics about the way metals cool and anneals (SUMAN; KUMAR, 2006; SINAEI; FATEMI, 2018). Commonly, SA provides good solutions (not necessarily the best ones), is simple to formulate, and can handle mixed problems of discrete and continuous space (SUMAN; KUMAR, 2006). An external parameter called *temperature* controls the evolution of the technique. SA assigns certain energy for each configuration. When an initial configuration is disturbed, the energy difference between the two states is responsible for the evolution of the system. When the new state is favorable (the energy decreases), then the new configuration is accepted. On the contrary, the new state is accepted or rejected according to a probability distribution, which is a function of *temperature* – when it is high, the probability of accepting an unfavorable state is greater (SINAEI; FATEMI, 2018). Starting from an initial solution, Simulated Annealing (SA) is a stochastic local search algorithm that iteratively explores the neighborhood of the current solution (FRANZIN; STüTZLE, 2019). SA always accepts improving settings. On the other hand, based on the *Temperature* parameter, SA takes worsening solutions using a probabilistic equation.

SA employs an analogy of the Monte Carlo integration for solving equations of state of physical systems composed of particles in statistical mechanics. At high temperatures, the particles are free to move, and the structure is subject to substantial changes. The temperature decreases over time, and so does the probability for a particle to move until the system reaches a state of lowest energy, its ground state (FRANZIN; STüTZLE, 2019).

(FRANZIN; STüTZLE, 2019) present a component view of the Simulated Annealing (SA) algorithm, according to Alg. 1. The SA includes problem-specific and algorithm-specific components, as follows.

The *two problem-specific* components are related to:

Algorithm 1 Component-based formulation of SA (FRANZIN; STüTZLE, 2019). The SA components are written in SMALLCAPS

Require: a problem instance π, a NEIGHBORHOOD $\mathbb{N}$ for the solutions, an INITIAL SOLUTION s_0, control parameters
Ensure: the best solution s* found during the search
1: best solution s* $\leftarrow incumbent\ solution$ ŝ $\leftarrow s_0$
2: $i \leftarrow 0$
3: $T_0 \leftarrow$ initialize temperature according to INITIAL TEMPERATURE
4: **while** STOPPING CRITERION is not met **do**
5: choose a solution s_{i+1} in the NEIGHBORHOOD of ŝ according to EXPLORATION CRITERION
6: **if** s_{i+1} meets ACCEPTANCE CRITERION **then**
7: ŝ $\leftarrow s_{i+1}$
8: **if** ŝ improves over s* **then**
9: s* $\leftarrow$ ŝ
10: **end if**
11: **end if**
12: **if** TEMPERATURE LENGTH is met **then**
13: update temperature according to COOLING SCHEME
14: **end if**
15: reset temperature according to TEMPERATURE RESTART scheme
16: $i \leftarrow i + 1$
17: **end while**
18: return s*

- the construction of an INITIAL SOLUTION, and
- the generation of a new candidate solution in the NEIGHBORHOOD

The *seven algorithm-specific* components that define an SA algorithm are:

- the choice of the INITIAL TEMPERATURE (line 3 of Algorithm);
- the STOPPING CRITERION, which determines when the execution is finished (line 4);
- the EXPLORATION CRITERION, which chooses a solution in the NEIGHBORHOOD (line 5);
- the ACCEPTANCE CRITERION, which determines whether the new solution replaces the incumbent one (line 6);
- the TEMPERATURE LENGTH, which indicates whether the temperature is updated (line 12);
- the COOLING SCHEME, which updates the temperature (line 13);
- the TEMPERATURE RESTART, the component responsible for resetting the temperature to its original or another high value (line 15).

2.3 System-level Simulation

Simulation is a way to evaluate the different solutions in a specific design. Modeling and simulation using a high level of abstraction perform an essential role during the early stages of development. Both allow the capture of the system's behavior and its interactions, usually requiring less modeling (fewer details to be modeled) and simulation efforts (ERBAS et al., 2007). Several authors advocate high levels of abstraction during Embedded Systems design: addressing its complexity (LEITE; WEHRMEISTER, 2016), favoring faster and cost-effective approaches to evaluate the design spaces (AN; GAMATIÉ; RUTTEN, 2015), and enabling estimates at early stages based on partial and uncertain information – pointing the design decisions in the correct direction (HENDRIKS et al., 2016).

Simulators with RTL (Register-Transfer Level) or cycle accuracy cannot meet the processing demand required in DSE activities, given the high volume of details to simulate and their prohibitive simulation time (HERRERA et al., 2014; GRÜTTNER et al., 2013; PALERMO; SILVANO; ZACCARIA, 2009). To overcome the problems related to simulation with cycle precision, we have simulators at the system level. According to Gries (GRIES, 2004), during a system-level simulation, the evaluation takes place at a high level of abstraction, being the system described by an interconnection of architectural blocks that represent the processors, memories, and buses. Thus, platform models or processing resources (processing elements - PEs) represent the system architecture (INDRUSIAK; DZIURZANSKI; SINGH, 2016). In the case of workload representation, coarse-grained models describe the applications, such as interaction processes or even whole procedures (GRIES, 2004). For example, task graphs are a common way of representing the workload (applications) in the context of embedded systems (INDRUSIAK; DZIURZANSKI; SINGH, 2016), as occur in (MIELE et al., 2015; BETEMPS et al., 2018). Other forms of representation include Kahn Process Networks (KPN) (ERBAS et al., 2007; SIGDEL et al., 2009b), UML/MARTE models (GRÜTTNER et al., 2013; HERRERA et al., 2014; LEITE; WEHRMEISTER, 2016), *dataflow* models (like *Homogeneous Synchronous DataFlow* - HSDF) (NOGUEIRA et al., 2016), Petri Nets (as *Colored Petri Net* - CPN) (CALLOU et al., 2011), automata (as *Timed Automata*) (JIANG et al., 2013; WANG et al., 2011), among others. These models represent the workload in the systems under analysis/construction.

Nowadays, more and more design exploration initiatives perform their tasks at the system level (GRIES, 2004; SINAEI; FATEMI, 2018), allowing decisions at early design phases. System-level Simulation (SIS) employs abstract models with coarse granularity to represent the architecture and the workload of the system under design. The employment of simulation and models at this level of abstraction, combined with the idea behind the early Design Space Exploration (DSE), directs the project towards the

most promising solutions and restricts the number of settings to be further evaluated in detail in lower levels of abstractions. Architectural and workload models are fundamental artifacts in the DSE process, mainly because the simulation allows us to test different solutions (architectures) in the execution of a specific workload (set of applications). SIS provides a viable path for evaluation of the design settings through DSE. Although SIS probably provides low precision results, its relevant information can assist designers in decision-making at early design stages. It can help the guiding of development initiatives into more detailed evaluations, preventing the assessment of not-so-promising settings.

The level of abstraction of the projects has been increased to the system level also to increase productivity, generating a great interest in TLM (*Transaction-Level Modeling*) (CAI; GAJSKI, 2003). TLM separates the details about communication from the computing components through the concept of channels. TLM increases the speed of simulation, allowing the exploration and validation of design alternatives using a higher level of abstraction (CAI; GAJSKI, 2003). Typically, TLM works in conjunction with SystemC (PATEL; SHUKLA, 2008). SystemC is a system-level design language with strong industrial support, being part of many industrial design flows. The use of C++ infrastructure and its object-oriented nature extends the usability of SystemC, being a suitable language for the co-simulation of hardware and software (PATEL; SHUKLA, 2008).

For System-level Simulation, high-level models of the system must be defined. Such models include aspects of workload and architecture. Considering heterogeneous systems, which allows different types of PEs, some modeling pattern is desirable to serve as a base. OpenCL provides such a programming model allowing to define host code (that runs usually in CPU) and device code, also known as an application kernel, that can run in diverse types of PEs in the exploiting of heterogeneity.

Fig. 6 shows a speed-accuracy trade-off spectrum, adapted from PIMENTEL (2017), of the levels of abstraction in simulating the processing components and the communication between the system components. Register-transfer level (RTL) is the lowest level of abstraction for simulating a digital system where the flow of digital signals between registers and combinational logic is explicitly simulated. Cycle-accurate simulation raises the level of abstraction by simulating the system components on a cycle-by-cycle basis. Both levels typically are too slow to perform full-system scale DSE. In the binary-translation technique, a simulation host computer executes an equivalent sequence of instructions translated from the target binary instructions. Besides, the host-compiled simulation uses a binary program directly compiled from the target program. In binary-translation and host-compiled simulation, the source code can be instrumented with timing and power consumption models according to the target architecture. When simulating communication by bus-cycle accurate simulation, the models

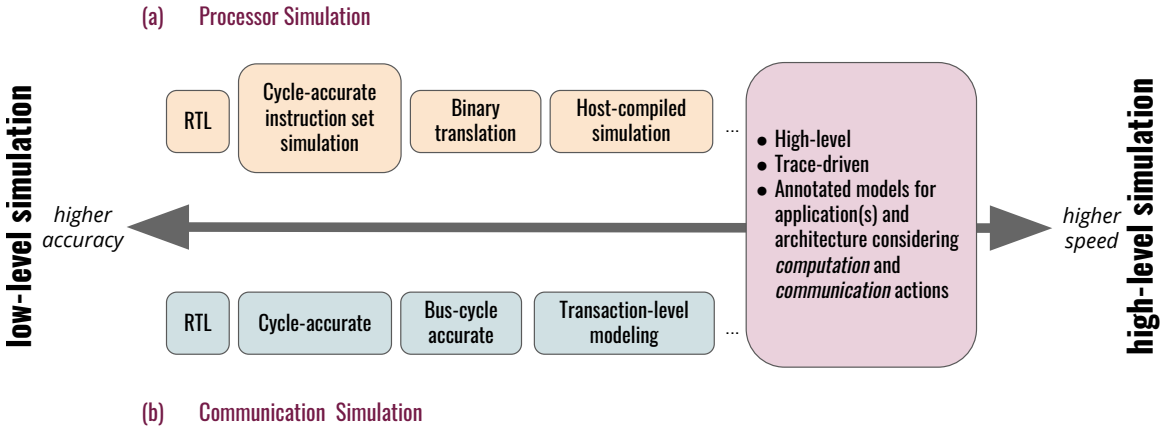


Figure 6 – Abstractions Levels of Simulation considering the Computation (a) – of the Processing Components – and the Communication (b) between the System Components (adapted from PIMENTEL (2017))

include all signals of the communication bus in a cycle-accurate fashion. Transaction-level Modeling (TLM) provides the highest level in the simulation of the communication between the system components. At this level, communication details in terms of signal and protocols are abstracted away by encapsulating entire transactions.

Still, according to PIMENTEL (2017), the previous techniques for processor simulation are all execution-driven simulation methods since we have the execution of a program (or a representation of it). However, there are alternatives for that. One of them is the trace-driven simulation in which the simulation is driven by event traces that have been collected through the execution of a program (or even other simulations or estimations of a program). To optimize the simulation speed, the trace events can represent computations and communications in a higher level of abstraction than machine instructions – e.g., execution of basic blocks or even whole functions. In a high-level simulation, an application model contains annotations describing the computation/communication actions at a coarse-grained level, typically considering the executions of entire functions. During the execution of the instrumented application model, it causes the generation of traces of events that drive the underlying architecture model, allowing to simulate the computation and communication events, also logging them in terms of non-functional requirements like performance and power consumption.

2.4 OpenCL

OpenCL (Open Computing Language)(KHRONOS, 2020) is an open, royalty-free standard for cross-platform and parallel programming for developing applications which *kernels* execute across a range of *device* types (KAELI et al., 2015). It allows the partition and programming of an application in two types of tasks (JÄÄSKELÄINEN et al., 2019):

(i) the tasks which deal with the input and output interaction and manage the execution (its implementation is generally called *host code*); and

(ii) the tasks which are the applications' core that can be accelerated by parallelization (the called *kernels*).

According to (KAELI et al., 2015), the OpenCL specification defines four models, as follows:

1. Platform model: it specifies that there is one processor coordinating the execution (the *host*) and one or more processors (the *devices*) executing the OpenCL kernels. It models an abstract HW model that directs the programmers while writing *kernels* (OpenCL functions);

2. Execution model: this model describes how to configure an OpenCL context on the host, preparing the host-device interaction, and describing a concurrency model to execute kernels on the devices;

3. Memory model: it abstracts a memory hierarchy to devices' utilization, regardless of the actual underlying memory architecture;

4. Programming model: describes how to map the concurrency model on the physical HW.

Fig. 7 shows the OpenCL platform model, defining an abstract architecture for devices. It defines a platform containing devices, each one organized as an array of functionally independent compute units. Compute Units are further subdivided into processing elements (KAELI et al., 2015). Considering the processor type for host and devices, often *host* code runs on the CPU. However, kernels execute on *devices* such as CPU, GPU, FPGA, DSP, etc., presenting an opportunity to exploit the system's heterogeneity. Thus, one can code SW using OpenCL (a high-level language) and set the application executing on general-purpose processors, such as CPUs and GPUs (considering the latter as General Purpose GPUs - GPGPUs), and even on HW accelerators such as FPGAs.

Further, the OpenCL Execution Model defines the execution environment, *host-device* interaction, and concurrency model for kernel configuring, i.e., how to decompose an algorithm in OpenCL work-items and work-groups. The unit of concurrent execution in OpenCL C is a work-item that executes the kernel function body. Instead of manually strip-mining the loop, a programmer often maps a single iteration of the loop to a work-item (KAELI et al., 2015). *NDRange* is an abstract *index space* defined by an N-tuple of integer. It specifies the dimension and the number of *kernel* instances (work-items) in the device (JÄÄSKELÄINEN et al., 2019). Each point in the index space allocates a work-item. Work-groups organize one or more work-items.

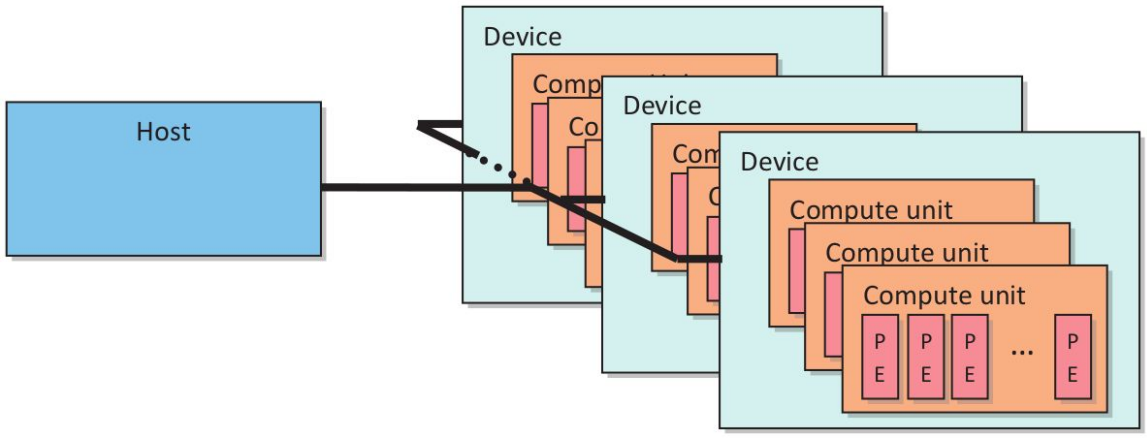


Figure 7 – OpenCL Platform Model (extracted from KAELI et al. (2015)).

Nowadays, OpenCL development environments, such as Intel FPGA SDK for OpenCL Software (INTEL, 2020a) and Xilinx SDAccel (XILINX, 2020), allows the employment of HW accelerators in computing systems. The kernels written in OpenCL enable a compilation that, through High-level Synthesis (HLS), creates an HW task able to run in a specific FPGA device. Moreover, through `pragma` directives, the compile/synthesis process can produce optimizations on the resulting modules according to the processing element – e.g., loop unrolling and replicated compute units targeting FPGA devices (ZOHOURI, 2018). Therefore, OpenCL provides a suitable programming model to explore heterogeneity and the performance/energetic efficiency of FPGA devices, also serving as a base modeling pattern in a system-level simulation context.

2.5 FPGA Devices as Hardware Accelerators

FPGA – Field Programmable Gate Array – is a configurable integrated circuit. It is customizable according to the target application(s) needs even after its manufacturing (SHATA; ELTEIR; EL-ZOGHABI, 2019). Two distinct layers compose the FPGA (Fig. 8): (*i*) configuration memory layer and (*ii*) HW logic layer (VIPIN; FAHMY, 2018). The *HW logic layer* contains computational resources, such as look-up tables (LUT), flip-flops, digital signal processors (DSPs), memory blocks, transceivers, etc. It also contains routing resources and switch boxes to connect the circuit components. The *configuration memory layer* stores the information describing the circuit, such as values in the LUT, memory block's initialization, routing information, flip-flop statuses, and others. A binary file stores the FPGA configuration information – usually called *bitstream*.

The complexity of FPGA HW description usually is the main hurdle in using it as an accelerator (MUSLIM et al., 2017) since it requires that the designer uses RTL (Register-Transfer Level) HW description languages such as Verilog or VHDL (SHATA; ELTEIR; EL-ZOGHABI, 2019). However, the release of environments like Intel FPGA

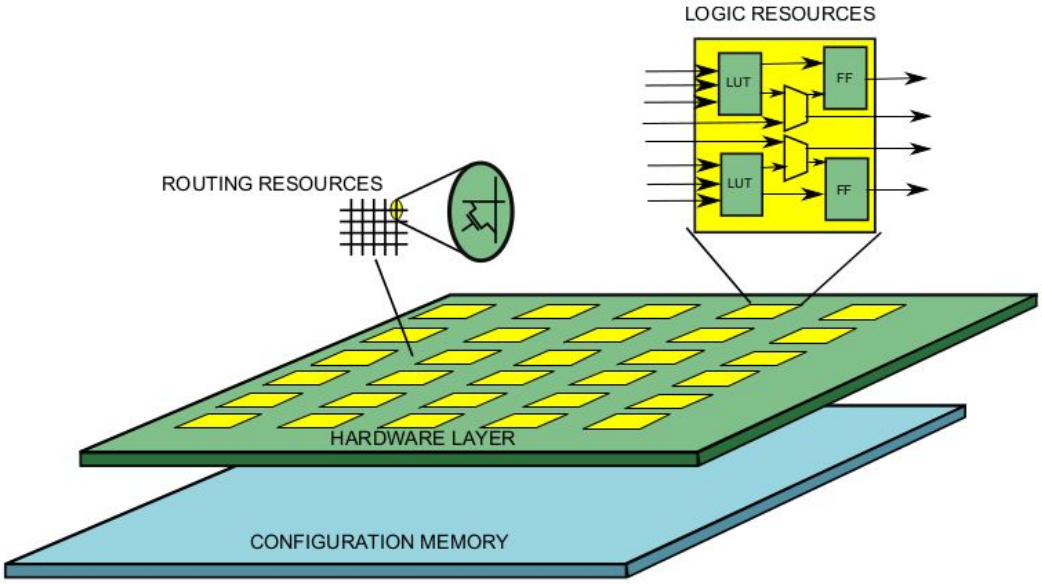


Figure 8 – FPGA Architecture – Layers (extracted from VIPIN; FAHMY (2018)).

SDK for OpenCL Software (INTEL, 2020a) and Xilinx SDAccel (XILINX, 2020) is changing this scenario, allowing the programming of kernels in OpenCL and, through High-level Synthesis (HLS), the using of FPGAs as accelerators. Besides, decreasing the development time and enabling portability of code for other devices, like CPUs and GPUs (SHATA; ELTEIR; EL-ZOGHABI, 2019).

Major FPGA vendors (Xilinx and Intel) provide devices with a key capability - Dynamic and Partial Reconfiguration (DPR) (VIPIN; FAHMY, 2018). DPR enables changes at run-time. Partial Reconfiguration (PR) allows modifying some portions of the device without stopping the remaining ones. Partially Reconfigurable Regions (PRRs) define one or more areas of the FPGA (PAPADIMITRIOU; DOLLAS; HAUCK, 2011) – denoting the earlier parts. Partially Reconfigurable Modules (PRMs) can share a PRR (PAPADIMITRIOU; DOLLAS; HAUCK, 2011). A partial bitstream configures the PRM logic into a PRR. DPR occurs on one PRR (or the full device) at a time, since normally there is a single PR controller. The smaller size of a partial bitstream allows a cheaper reconfiguration time. Frequently, designers use a blank bitstream as a strategy to reduce power consumption. A blank bitstream configures an empty PRR (even for the full device) that clears the configuration and allows power saving (BONAMY et al., 2014) (PAPADIMITRIOU; DOLLAS; HAUCK, 2011).

PR capabilities provide some advantages (VIPIN; FAHMY, 2018) (BONAMY et al., 2014), such as: increasing the device's logic density, reducing the reconfiguration time, decreasing the device's size and its static power, and online configuration according to adaptive applications. A drawback happens when a necessary PRM is not available on its PRR – in this case, a PRR reconfiguration will take place consuming time and power. A sole FPGA device can be deployed as several *Virtual Hardware Accelerators* (*Vir-*

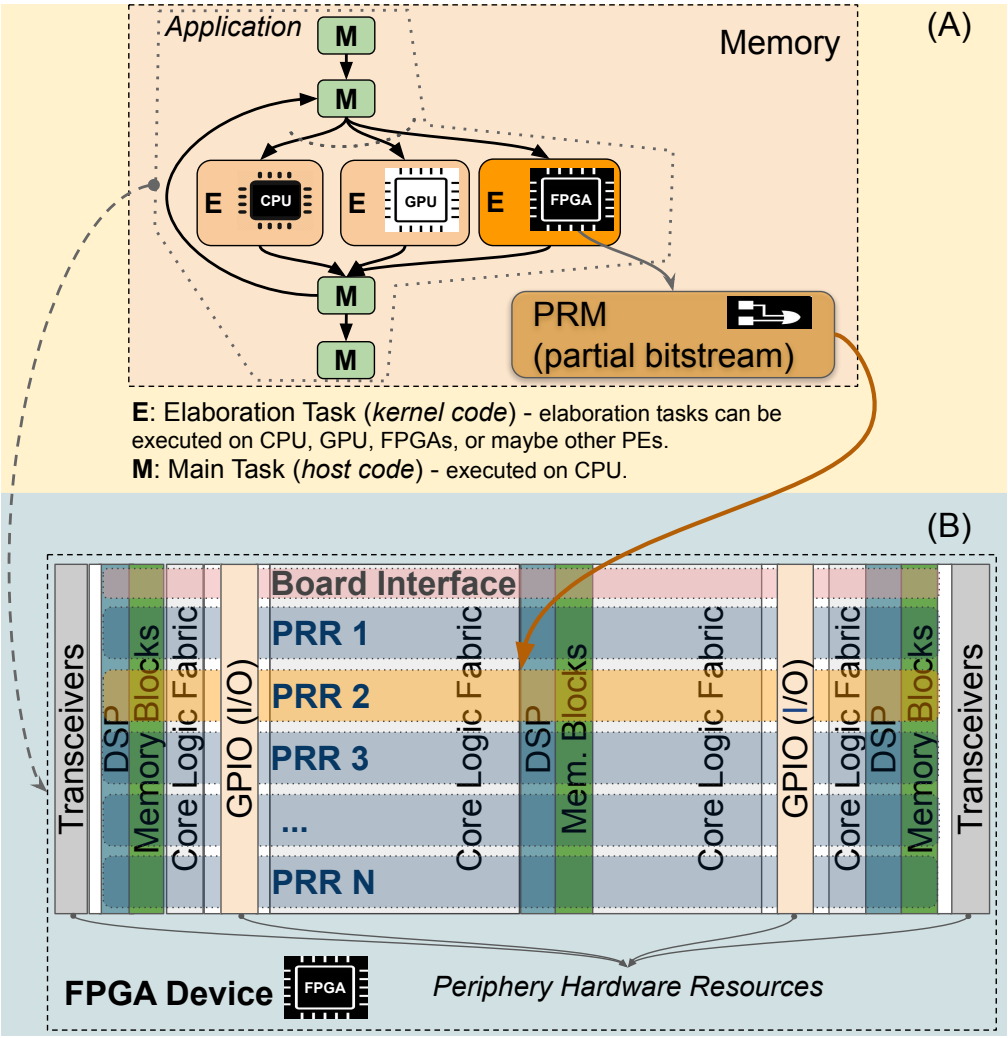


Figure 9 – PRRs, PRMs, board interface, and periphery HW. (A) Application model and partial bitstream. (B) FPGA device model.

tualHWAccel)s since each PRR is a potential processor. There is a trade-off between the number of PRRs and their size (granularity), contrasting with the implementation of the PRM and its kernels.

Fig. 9 illustrates the presented concepts. PRRs splits the FPGA device into many portions that can contain the implementation (programmed by a partial bitstream) of a PRM. Periphery HW resources are part of the FPGA device – e.g., transceivers and I/O components, but not part of a PRR. The board interface region maintains the platform interface logic – it is a static region (together with the periphery HW) normally configured only in an initial full configuration process. A PRR contains core-only resources, such as logic fabric, DSPs, and memory blocks.

2.6 Chapter Summary

This chapter has presented background topics associated with the thesis theme. Since we deal with computing systems composed of HW and SW components, Embedded Systems (ES) is a representative concept. A designer needs to understand the design steps of an ES, perceiving the importance of evaluating the HW/SW interface and their co-design. Design Space Exploration (DSE) is a method to assess the design points, usually through heuristics. DSE demands an evaluation scheme for the design points. System-level Simulation (SlS) allows such an evaluation by handling high-level models of the system and simulating its behavior, providing performance metrics. OpenCL gives the model basis from which we describe the applications and architecture. Based on OpenCL HLS tools, we handle kernel code. It allows generating HW task implementations for an FPGA as well as performance and power estimations. In the architecture, an FPGA may play a customizable reconfigurable HW accelerator role. It can also be capable of PRR partitioning, employ the DPR feature, and implement different design optimizations of HW tasks.

3 RELATED WORKS AND RESEARCH CHALLENGES

Several works have treated themes related to system-level modeling and simulation, reconfigurable architectures, and design space exploration (DSE). The design of heterogeneous systems encharges designers with many issues to deal with during the development initiatives. Moreover, the employment of different kinds of processing elements (PE) enlarges the system's design space. Custom PE (e.g., HW accelerators) augments this scenario. Even more, if some reconfigurability is present. Regarding this type of flexibility, FPGA devices are in the front line due to their custom and reconfiguring features, beyond providing performance and power efficiency.

Regarding the design space, DSE is a suitable method to evaluate candidate solutions (architecture configurations) for a computing system given a specific workload (applications). Moreover, early DSE is of paramount importance since the decisions made at this moment may influence all the following design directions. However, during the first design steps, when possibly no real prototype is available, the designers need method alternatives to evaluate a system under consideration, notwithstanding if it doesn't even exist.

System-level modeling and simulation is a way to describe and evaluate a system's architecture and workload – as a virtual platform. Regarding simulation, there is a trade-off between accuracy and simulation performance. More detailed models diminish the performance whereas increasing accuracy. During early evaluations, there is no much time to create detailed models. Hence, usually, practitioners employ high-level system models describing and simulating it.

Considering methodologies, frameworks, simulators, and platforms for the DSE of heterogeneous systems featuring HW accelerators and reconfigurability, several works have been presented dealing with system design and DSE issues in this scenario.

The performance and power efficiency of HW accelerators (e.g., an FPGA) make this sort of PE an interesting choice to include in the system's design. Different abstraction levels describe the application's (HW) tasks that run on such PE. Hardware Description Languages (HDL) provide a low-level representation for HW tasks, normally HLS-obtained from C-like code. Works like LIANG et al. (2017); FENG et al.

(2017); HUANG et al. (2019); LIGNATI et al. (2021) employ this representation level. Other works employ high-level models or languages to describe HW tasks' functionality. Tasks graphs (MIELE et al., 2015), Kahn Process Networks (KPN) (SIGDEL et al., 2009b; PIMENTEL; ERBAS; POLSTRA, 2006), Control and Data Flow Graph (CDFG) (DUHEM et al., 2015), Homogeneous Synchronous DataFlow (HSDF) (NOGUEIRA et al., 2016), Abstract Clock Models (AN; GAMATIÉ; RUTTEN, 2015), AADL (Architecture Analysis and Design Language) descriptions (BLOUIN et al., 2011), UML (Unified Modeling Language) (HUANG; HSIUNG; SHEN, 2010) and UML/MARTE (Modeling and Analysis of Real-Time and Embedded systems) (GRÜTTNER et al., 2013; HERRERA et al., 2014; QUADRI et al., 2010) models are examples of it. Regarding the architecture, the approaches usually model its elements in a generic way augmented with profiled information – e.g., power, frequency, logic elements (area), and others. Further, some works deal directly with HDL-alike code in standard HW design tools.

When simulating a (virtual) platform, the works employ many different levels. We can observe simulation's abstractions since RTL in standard simulators (QUADRI et al., 2010; HUANG et al., 2019) and academic ones (Verilator) (FENG et al., 2017; LIANG et al., 2017), to pre-RTL (represented by Dynamic Data Dependence Graphs – DDDG) in SHAO et al. (2016), MAKNI et al. (2018), ZHONG et al. (2016), and ZHAO et al. (2020). However, higher abstraction levels are preferred in other works using different model representations implemented, for example, in SystemC (MIELE et al., 2015; DUHEM et al., 2015; GRÜTTNER et al., 2013; HERRERA et al., 2014; BRITO et al., 2007b). Other approaches simulate its high-level models using specific tools and formalisms such as DEVS (NOGUEIRA et al., 2016), rSesame (SIGDEL et al., 2009b), CLASSY (AN; GAMATIÉ; RUTTEN, 2015), OSATE (BLOUIN et al., 2011), Rhapsody (HUANG; HSIUNG; SHEN, 2010), and SCOPE+ (HERRERA et al., 2014).

Regarding the system's architectures in the related works, HW accelerators (usually implemented through FPGA devices) are the mainstream processing unit for dealing with the application's kernels. Considering the FPGA's features, the research initiatives present their claims based on features like reconfigurability and DPR. The main goals of the research projects usually include DSE activities and their outputs, and the generation of design artifacts and code. However, these are not the only themes. Some works put their efforts on reconfiguration issues such as bug detection (GONG; DIESSEL, 2014) even as the DPR/PRR modeling and design (BRITO et al., 2007b; QUADRI et al., 2010; DUHEM et al., 2015). Others focus on the design of specific domains such Network Security Systems (HUANG; HSIUNG; SHEN, 2010), Multimedia Embedded Systems (NOGUEIRA et al., 2016), Resource Management (MIELE et al., 2015), FPGA Performance|Power|Area Estimations (MAKNI et al., 2018; ZHONG et al., 2016; ZHAO et al., 2020), and CPU-FPGA-based Cloud System Resource Provisioning (LIGNATI et al., 2021). Evaluation of the interactions between the system's HW ac-

celerated PE and memory system is also a frequent study theme (SHAO et al., 2016; LIANG et al., 2017; FENG et al., 2017). DSE completes the works' main goal list, with efforts in SoC evaluation (AN; GAMATIÉ; RUTTEN, 2015; BLOUIN et al., 2011; HUANG et al., 2019), HW/SW co-design (GRÜTTNER et al., 2013; HERRERA et al., 2014), and task mapping (SIGDEL et al., 2009b, 2012).

The works in the area use different abstraction-levels during the simulations, also employing diverse representations for the workload and architecture itself. Moreover, some works deal with energy/power perspectives but being a gap in others. FPGA units figure in the works as a suitable and efficient processing element (PE). However, key features of reconfigurable HW are not fully applied – like multiple Partially Reconfigurable Regions (PRRs) and the utilization of Dynamic and Partial Reconfiguration (DPR) within the FPGA's portions (the PRRs). Thus, in the following sections, we discuss the related works' main characteristics organizing them based on these raised aspects.

3.1 Related Works

The related works deal with different levels to perform simulations. Also, there is a dichotomy related to a power model as well as for DPR and PRR features. Thus, in the next sections, we organize them according to these characteristics.

3.1.1 High-Level Simulation

In this section, the presented works employ high-level modeling in describing the system's components. The authors utilize some models representing the applications and the processing elements during the simulations.

3.1.1.1 Works with Power Model

AN; GAMATIÉ; RUTTEN (2015) presents a modeling and analysis framework for DSE of Adaptive applications on MPSoCs. It is a clock-based approach that describes SW and HW exploiting the notion of abstract clock models borrowed from synchronous dataflow languages. The framework uses the CLASSY (CLock AnalysiS SYstem) tool (Fig. 10 (A)) for modeling, scheduling, and analysis (simulation). Moreover, the framework includes a DSE module (implemented using the evolutionary algorithm NSGA-II). The approach uses pre-profiled information to annotate the model elements about execution latency and energy consumption, using probability distribution in the interval best-worst cases regarding latency. Event occurrences and their precedence relations represent an application. A set of static behaviors capture applications' behavior integrating dynamism by an associated controller (usually, a finite state machine – FSM) that dictates the sequence of static behaviors along time. Fig. 10 (B) illustrates two

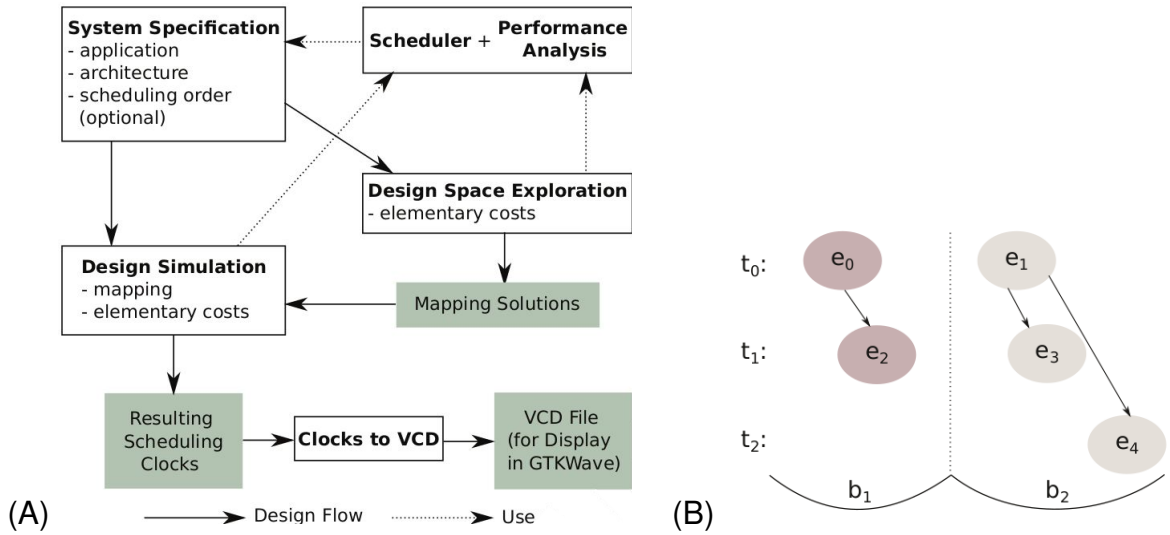


Figure 10 – Overview of the CLASSY Tool (AN; GAMATIÉ; RUTTEN, 2015)

behaviors b_1 and b_2 – events e_0 and e_1 belong to task t_0; events e_2 and e_3 belong to task t_1; and event e_4 belongs to task t_2. DPR feature is available, but reconfiguration time/power is not taken into account. The work does not use multiple PRR within the device's reconfigurable zone. Besides, the approach doesn't evaluate area aspects.

In BLOUIN et al. (2011), the authors present an extension of AADL (Architecture Analysis and Design Language) intending to model reconfiguration features of a platform. It regards the reconfigurable logic and power requirements within Multiprocessor Reconfigurable System-on-Chip (MPRSoC). The reconfigurable resource is typically an FPGA embedded in the MPSoC (known as eFPGA). The authors describe a multi-layer approach using the AADL extension, with layers for describe:

- (i) the general characteristics of an FPGA device (Fig. 11 (a));
- (ii) the FPGA static (non-configurable) part of a specific device (Fig. 11 (b)); and
- (iii) the HW task allocation part – the actual use of the FPGA – where IP blocks configure its region during the execution.

Thus, AADL descriptions define the static part – including IPs for generic processors and buses – and the configurable one – depicting elements such as memory, buses, IP blocks, and ports. Next, the application's descriptions delineate its basic structure containing threads (including for the controller) and aspects related to the deployment such as application, platform, connections, and bindings. An eclipse IDE-based toolchain provides the infrastructure to deal with AADL language (OSATE tool – Open Source AADL Tool Environment), requirements definition and analysis language (RDALTE tool), and graphical editor (ADELE AADL). Requirements expressed in OCL (Object Constraint Language) are attached to the models. It allows the tools to evaluate (and trace) if the architecture does not meet some requirements. Although dealing

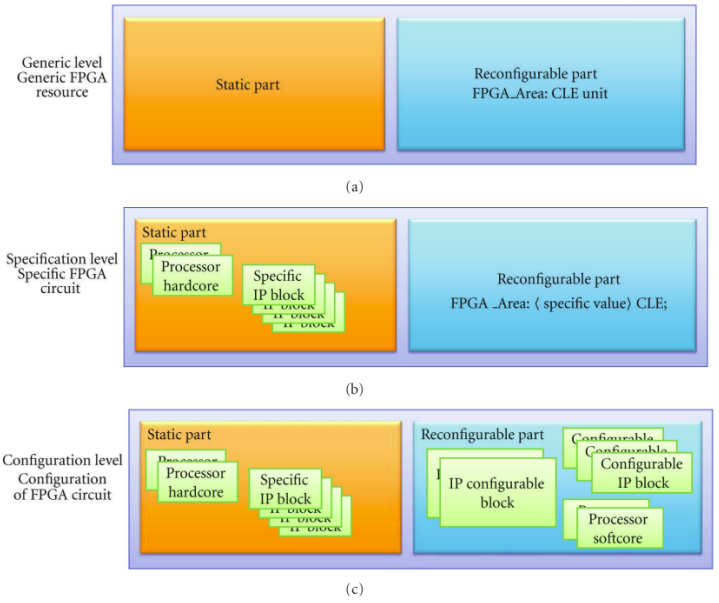


Figure 11 – Multilayer Approach using the AADL Extension (BLOUIN et al., 2011)

with DPR, multiple PRR definition is not part of the AADL extension. Moreover, the approach does not model the reconfiguration time during an HW task configuration into the configurable region.

MIELE et al. (2015) propose the SAVE system-level simulation framework implemented in SystemC and TLM that aims to validate runtime resource management policies for Heterogeneous System Architectures. The framework intends to deal with runtime management to allocate system resources to applications. For that, it uses data about PE's efficiency. Besides, it aims to fulfill Service Level Agreement while enforcing system-level constraints. Task graphs represent the applications encompassing data about task types and latencies, performance counters, number of threads, among others. Still, the architecture is modeled as a set of generic resources, describing performance and power models. The model's parameters can be extracted through simulation, execution in real HW, or even estimated by an experienced developer. Task graphs model the applications (Fig. 12 (A)). It includes main tasks (usually executed on CPUs) and elaboration tasks. CPUs, GPUs, and HW Accelerators are the possible processors to accelerate an elaboration task. Edges interconnect the tasks representing the control dependencies between them. Regarding the edges, it is worth mentioning the branch edges. These edges indicate all alternative implementations for the applications' kernel(s).

The SAVE simulation framework contains SW and HW components (Fig. 12 (B)). Processing units (e.g., CPUs, GPUs, and HW Accelerator) components represent the system's processors that communicate with the management (SW) components via a communication channel (the latter represents a virtual communication and memory infrastructure). A workload generator represents the user actions with the system

transmitting the workload to the System-calls Emulator. This emulator is in charge to initiate/manage the applications, imitating the OS behavior. The Governor component collects information from the system components, also serving as a stub where the designer can implement the resource management policies under investigation. The SAVE framework uses high-level models to describe the (virtual) platforms and deals with performance, power, and utilization metrics. Thus, it is suitable to employ in the early phases of a design initiative. However, although the authors list an HW accelerator in the system's processors, it is not available in the framework implementation. Moreover, there are no mentions of reconfiguration features (e.g., DPR and PRR) related to HW accelerator such as FPGA. In a partnership with the SAVE authors, we have accessed the SAVE's implementation code. We use this framework as a basis to build `SAVE-htlp` (BETEMPS et al., 2018) – a framework (Sec. 4.2) capable of heterogeneous task-level parallelism between its processing elements during a platform simulation of applications' kernels. Additionally, we build our simulation infrastructure (FEHetSS) on top of `SAVE-htlp` – as described in Sec. 4.4.

In NOGUEIRA et al. (2016), the authors present a DSE approach for the performance evaluation of Multimedia Embedded Systems. Also, they propose a multi-objective algorithm for system-level design explorations. In the system descriptions, HSDF (Homogeneous Synchronous DataFlow) graphs represent the applications as an ATG (Architecture Template Graph) directed graph describes the hardware platform. A timing requirement (deadline) annotates each application task. On the other hand, architectural elements have a set of values featuring their price, power, and idle power. A GA (Genetic Algorithm) based DSE experiments the mapping alternatives between applications and architecture resources during its processing. The approach applies the Parallel DEVS (Discrete EVent system Specification) formalism enabling the modeling, analysis, and simulation. During the DSE, probability distributions describe the task execution times and their inter-arrivals times. Fig. 13 describes the methodology of the work. In short, GA heuristic and stochastic simulation lead the DSE aiming to find optimal (or near) Pareto solutions. The formalism in its usage demands developers with specific expertise. High-level models describe the system. However, the approach does not take advantage of reconfigurable HW flexibility through reconfiguration features (DPR and PRR).

The COMPLEX framework and methodology are described in GRÜTTNER et al. (2013) and HERRERA et al. (2014), respectively. GRÜTTNER et al. (2013) describes an approach of Platform-based Design that uses system-level time and power estimation to perform the DSE detaching the use of UML/MARTE models as a design input with the automatic generation of an executable SystemC model. According to Fig. 14, COMPLEX is a reference framework that incorporates standard UML and their profile MARTE. These models describe the architecture and workload elements

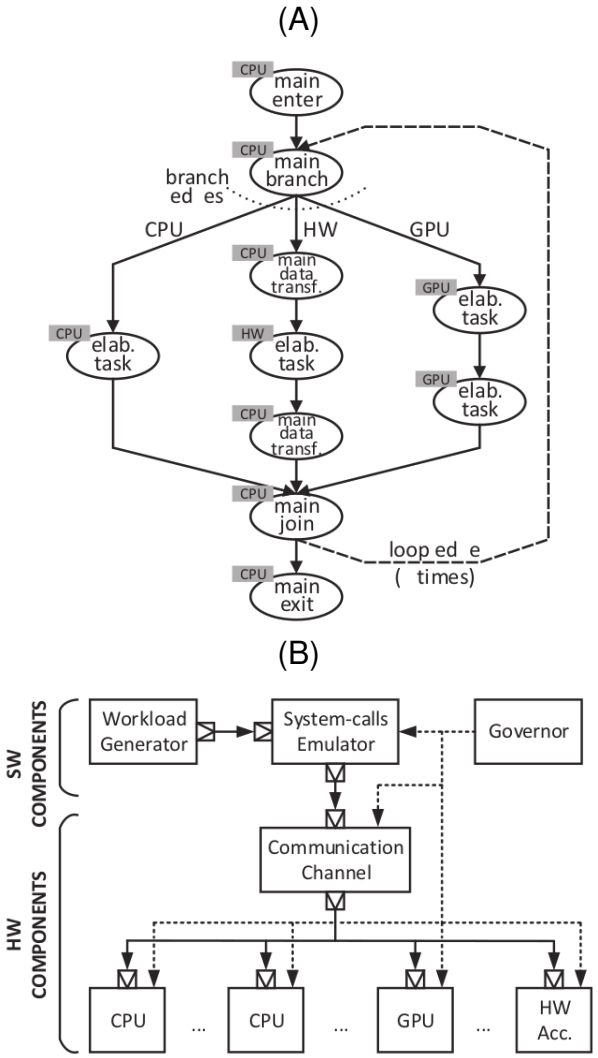


Figure 12 – SAVE Framework: (A) Application Task Graphs. (B) Simulator Structure in SystemC/TLM (MIELE et al., 2015)

of a virtual system. Using Model-Driven Engineering (MDE), an application model, a system input stimuli, and an architecture description pass through a chain of models' processing/transformations. It generates an executable prototype in SystemC containing timing and power annotations inserted from Custom HW and SW estimations and a library with pre-existing IP/virtual components. In HERRERA et al. (2014), the authors describe a DSE approach that includes MDE and Electronic System Level (ESL) technologies. The framework capture a set of solutions based on UML/MARTE models and functional code. It produces an executable, configurable, and high-performance model. Then, SCoPE+ tool (HERRERA et al., 2014) allows the simulation of different solutions and generates performance estimates. SCOPE+ includes a performance estimations library containing different architectural mappings. The framework demands several system viewpoints as models describing data, functional, communication&concurrency, platform, architecture, and verification aspects. Thus, many complex models are necessary to employ the approach at early design phases. Besides, the

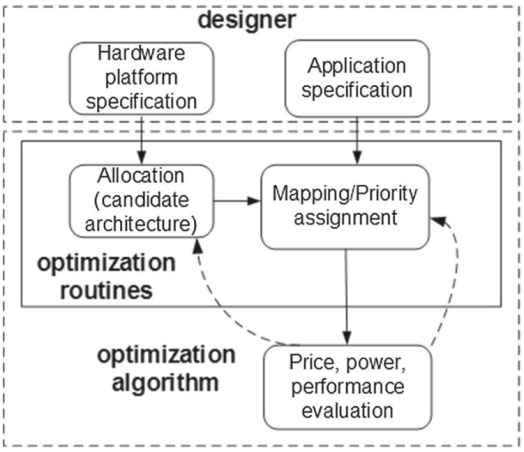


Figure 13 – The Proposed Method for GA-based DSE Using Parallel DEVS Formalism to Evaluate the Candidate Solutions (NOGUEIRA et al., 2016)

employment of custom HW in the virtual system does not include DPR/PRR aspects.

3.1.1.2 Works with No Power Model

DUHEM et al. (2015) present FoRTReSS, a flow that automates the HW DSE inferring a set of reconfigurable regions from task resources information. FoRTReSS relies on a SystemC simulator, RecoSim, that allows the designer to develop and evaluate its scheduling algorithms. Fig. 15 presents an overview of the FoRTReSS flow. It follows the Y-Chart approach describing the applications through control data flow graphs (CDFGs) where each task has some timing characteristics (such as deadline) and a set of possible implementations (with different performance, resources, and energy tradeoffs). The DSE maps tasks to HW (onto a PRR) or SW (in a processor core). The system architecture describes an FPGA and a set of processor cores. FoRTReSS employs the application resource requirements and FPGA description to find potential reconfigurable regions (PRRs). RecoSim validates the application's quality-of-service, generating traces, statistics, and log files for every simulation. The output of the FoRTReSS flow is an architecture fully defined in terms of PRR, including floorplan descriptions. FoRTReSS does not deal with energy and power aspects in the DSE (such as energy minimization), only planned for future versions of the flow. Besides, FoRTReSS might require full netlists of each HW task implementation, demanding low-level descriptions. Thus, the main focus of FoRTReSS is about the meeting of real-time constraints producing PRR descriptions embracing low-level aspects such as floor planning.

Sesame (SIGDEL et al., 2009a) and rSesame (SIGDEL et al., 2009b) employ a discrete-event (trace-based) simulation to evaluate applications in the form of Kahn Process Networks (KPNs). In rSesame, Sesame is augmented to model and simulate the dynamic reconfigurable behavior of the Molen dynamic reconfigurable architecture. Three types of tasks are possible in rSesame:

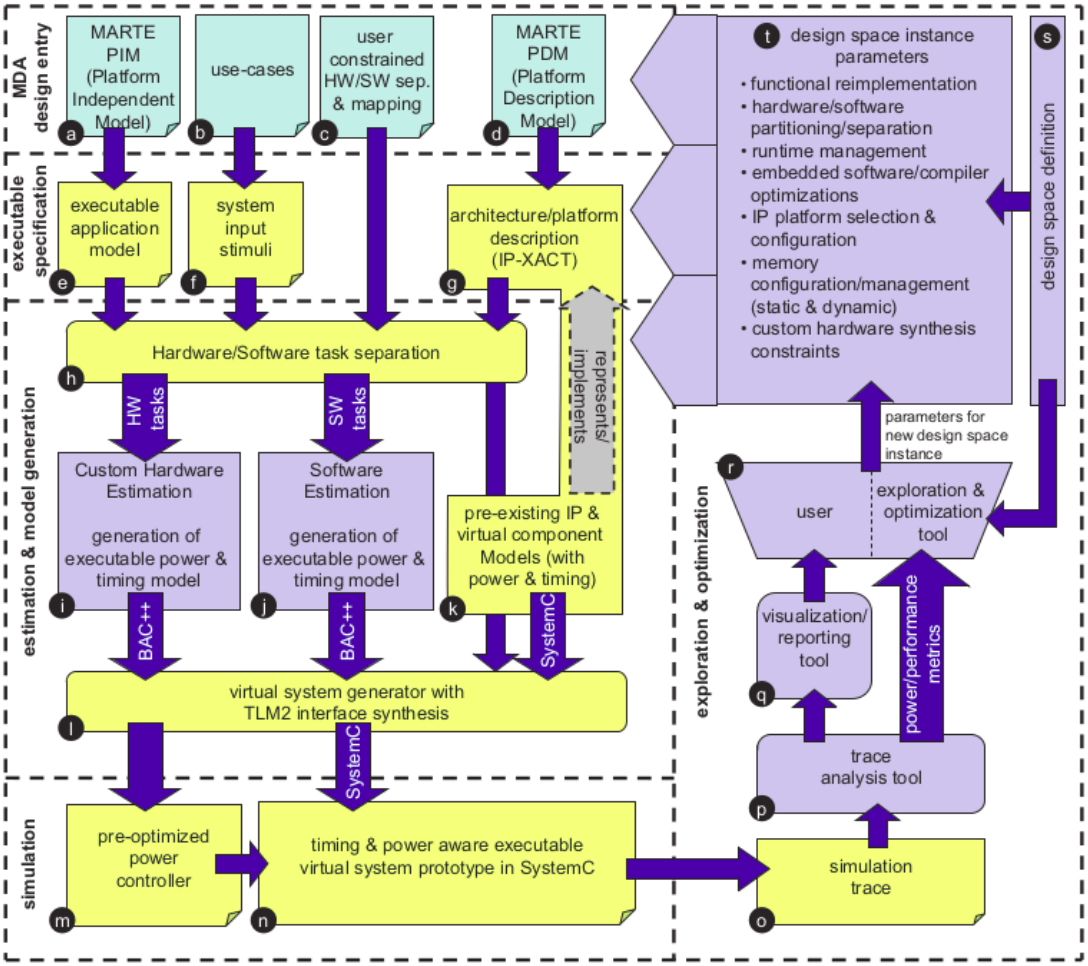


Figure 14 – The COMPLEX Reference Framework (GRÜTTNER et al., 2013)

- (*i*) SW Tasks: General Purpose Processor (GPP), such as CPU, executes it;
- (*ii*) HW tasks: it is executed on Reconfigurable Processors (RP), such as FPGAs; and
- (*iii*) Pageable tasks: this type of task can switch between the two types of processors.

Fig. 16 shows the three layers of the rSesame architecture model. The application layer contains the KPN model. On it, each task maps to a virtual processor in the mapping layer. The Runtime Mapping Manager makes decisions about the processors' allocation. GPP and RP form the architecture layer where a resource manager maintains information that feeds the mapping manager. The RP may contain several CCU (custom computing units). Each CCU can process events only after its configuration – as occurs with PRR. SIGDEL et al. (2012) presents an evaluation of task mapping heuristics. It employs rSesame as a Modeling/Simulation framework employing the following metrics: execution time, percentage of HW and SW usage, number of reconfigurations, time-weighted area usage, and reusability efficiency. Although the

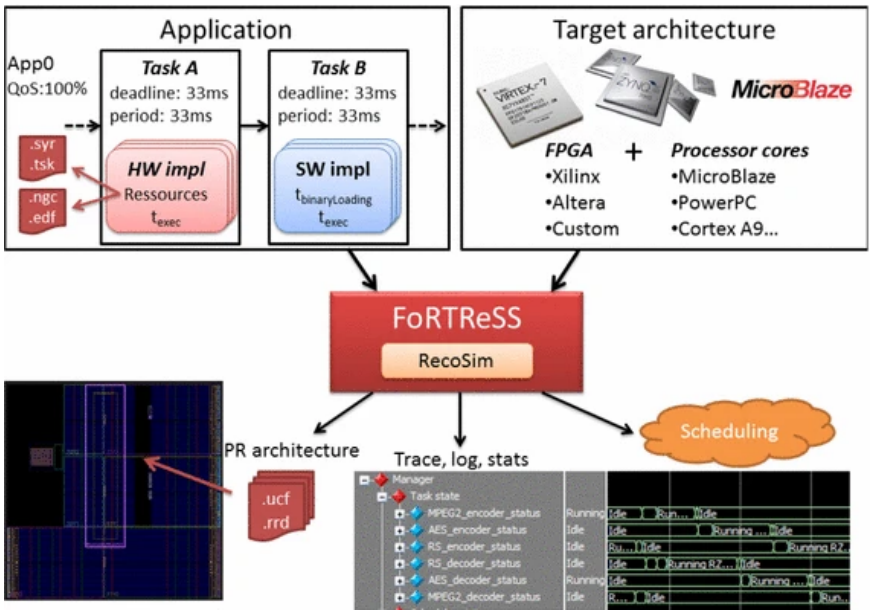


Figure 15 – FoRTReSS Overview (DUHEM et al., 2015).

rSesame environment deal with high-level representations of the system and applications, it does not present power or energy models for the architectural elements.

3.1.2 Low-Level Simulation

In this classification, the works usually employ RTL-level, Pre-RTL, cycle-accurate simulation, or even direct interaction with HW devices to evaluate the system performance. Pre-RTL is a dataflow representation obtained by a series of transformations from C code to Dynamic Data Dependence Graph (DDDG) (SHAO et al., 2014). Thus, the applications' description may be at a high level but ends up in a low-level representation aiming simulation or even the applying of analytical models.

3.1.2.1 Works with Power Model

HUANG; HSIUNG; SHEN (2010) presents the UML-based Co-design Platform (UCoP). Dynamically Partially Reconfigurable Network Security Systems (DPRNSS) are the target applications. The authors advocate the employment of UML modeling&simulation (through Rhapsody UML tool) integrating a specific FPGA device (Xilinx Virtex II) in the platform. According to Fig. 17, UCoP integrates FPGA-platform specific libraries (FUSE APIs and PCI drivers) into the Rhapsody tool for file generation aiming simulations and accurate estimations. The dynamic FPGA area implements two PRR (re)configuring the applications' HW functionalities – specifically cryptographic and hash designs. Three categories classify the UML models: (i) SW application, (ii) HW configuration, and (iii) system management. Through a partially reconfigurable HW task template, designers can focus on their HW designs. The direct interaction with real HW provides the system estimations. Indeed, the work does not employ sim-

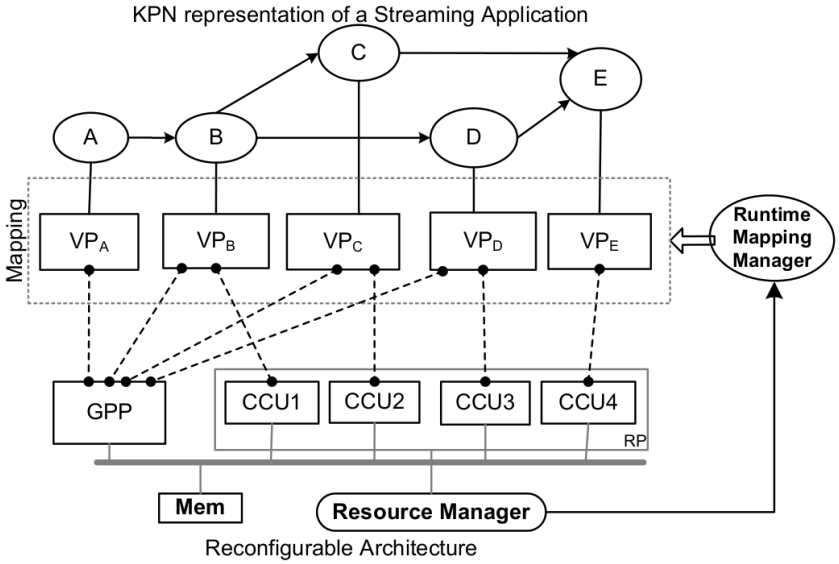


Figure 16 – rSesame Model for a Generic Reconfigurable Architecture (SIGDEL et al., 2009b)

ulation, but a direct HW interaction. Albeit UCoP uses UML models to describe the system's aspects providing accurate estimations, it is directed to a specific device and application domain, restricting their use.

LIGNATI et al. (2021) present MultiVers, a framework that exploits the automatic generation from HLS to build libraries containing multiple versions of each incoming kernel request. It implements an automatic resource provisioning solution for CPU-FPGA Cloud Environments by allowing cloud providers to prioritize either parallelism (low kernel area requirements), energy, or performance at run-time, according to its requirements and workload at the moment. According to Fig. 18, the framework's flow contains five steps: (i) it collects the kernel requests; passing to the (ii) *kernel selection* based on the cloud provider's optimization goal, represented as an optimization tuple regarding the kernel's area, performance, and energy; the (iii) *batch generation* step sends the versions of the previously selected kernel to a FIFO structure; passing to an (iv) *allocation strategy* based on a collaborative allocation heuristic which generates a collaborative solution containing a set of kernel arrangements over time, defining the FPGA configuration (also considering its reconfiguration) and the CPU(s) allocation; the (v) *execution* phase fires the kernels' execution in the CPU-FPGA environment. This work considers the PRR size and evaluates the necessary reconfigurations during its experiments aiming to minimize them. Thus, MultiVers employs PRRs and DPR features, even not explicitly defining multiple PRRs.

In gem5-Aladdin (SHAO et al., 2016), the integration of gem5 with Aladdin simulator (SHAO et al., 2014) deals with System on Chip (SoC) models. As shown in Fig. 19 (B), these SoC models can encompass CPU cores and their memory components and Accelerators with different memory arrangements like Cache and Scratchpad Acceler-

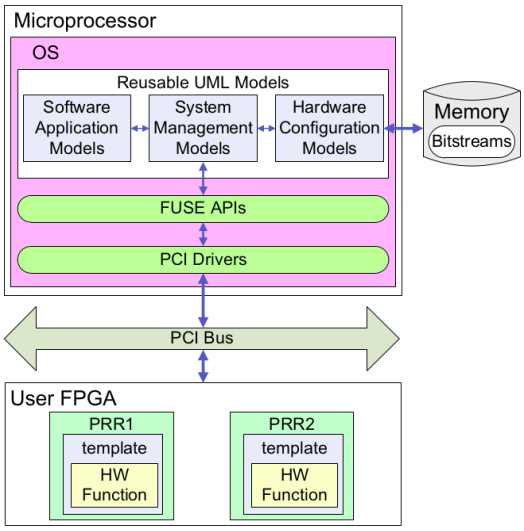


Figure 17 – UCoP UML-based HW/SW Co-design Platform (HUANG; HSIUNG; SHEN, 2010)

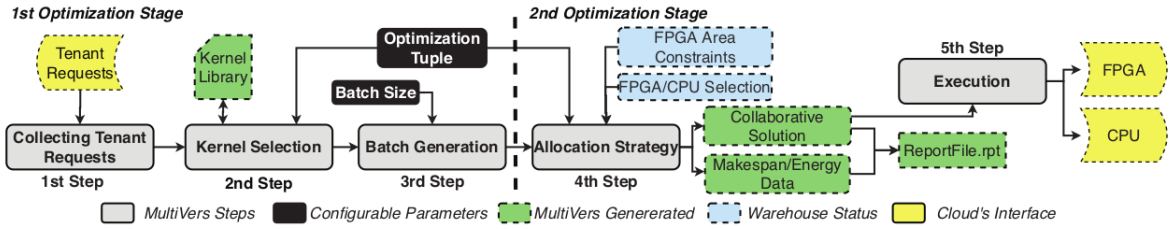


Figure 18 – MultiVers Framework Overview (LIGNATI et al., 2021)

ator. According to the authors, ”accelerator” refers to an application-specific hardware block. Thus, we can embrace this term as including FPGA devices. Fig. 19 (A) shows the Aladdin framework in all its phases and artifacts. Aladdin (SHAO et al., 2014) is a pre-RTL, power-performance simulator. It takes high-level descriptions (C code) of the applications transforming them into a dynamic data dependence graph (DDDG) to describe the accelerator's code. DDDG is a dataflow description generated in a series of transformations passing through an intermediate (and idealistic) representation to a program and resource-constrained one. Besides, based on activity traces and power characterizations of DDDG nodes (e.g., multipliers, adders, shifters, etc.) and registers, the framework outcomes a power-performance accelerators model. gem5-Aladdin allows evaluating the accelerator's interactions with the other system components, like CPU cores and memory settings. Albeit gem5-Aladdin deals with high-level models, it does not consider DPR aspects, missing opportunities to take advantage of the inherent flexibility of reconfigurable devices like FPGAs.

MAKNI et al. (2018) present the framework HAPE – High-level Area and Power Estimation – which employ high-level analytic models for the area and power estimations aiming at accelerators based on FPGA devices. It uses a pre-RTL representation, a DDDG, generated from an LLVM (Low-Level Virtual Machine) intermediate repre-

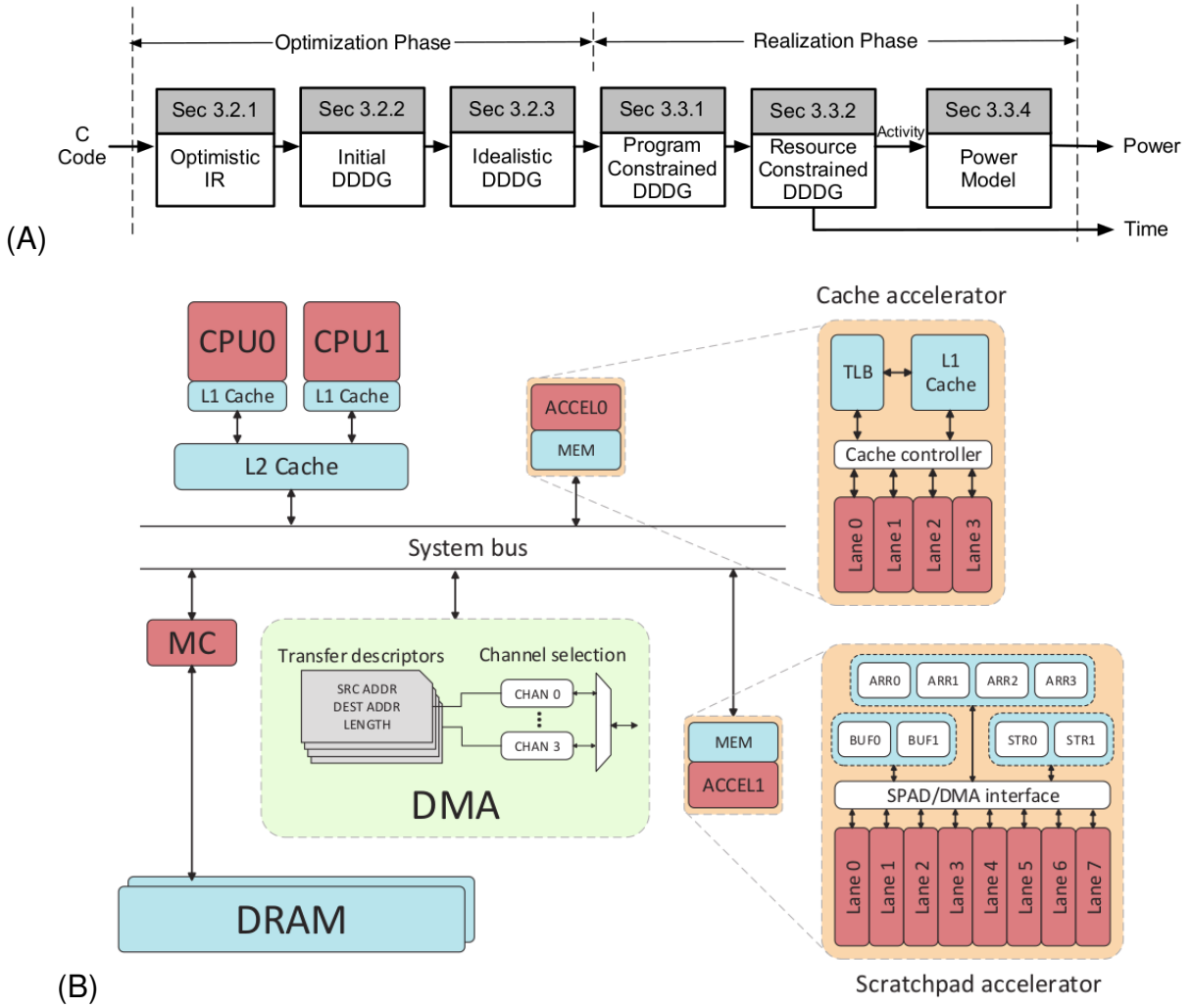


Figure 19 – (A) The Aladdin Framework Overview (SHAO et al., 2014). (B) A SoC Architecture Sample in gem5-Aladdin (SHAO et al., 2016).

sentation (IR) which is obtained from a dynamic execution trace to represent program behaviors. The framework produces the IR during an optimization phase, using the dynamic execution trace and optimization pragmas – the latter is inserted in the high-level specification of the program (in C/C++). During a generation phase, the framework produces an optimized DDDG. Based on the DDDG representation and in FPGA resource constraints, HAPE uses its analytical models to produce a Performance-Area-Power estimation. HAPE does not explore features related to DPR and PRR.

3.1.2.2 Works with No Power Model

ReSim (GONG; DIESSEL, 2014) is a cycle-accurate simulator that assists designers in detecting fabric-independent bugs. Its designs encompass dynamic reconfiguration. This simulator aims to deal with the reconfiguration machinery. It uses simulation-only bitstreams to model two dynamic reconfiguration characteristics – the bitstream traffic and the bitstream content. The focus of ReSim is to provide the apparatus to

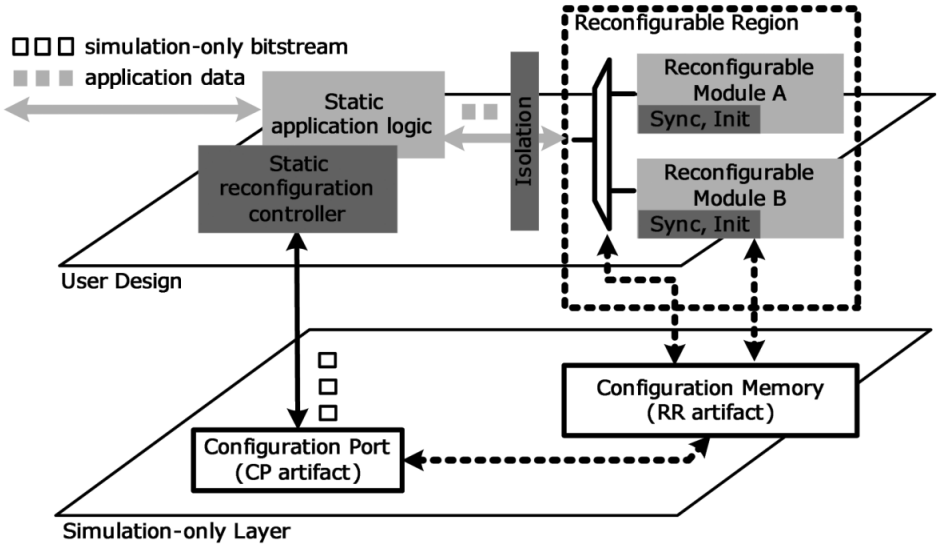


Figure 20 – ReSim Simulation Artifacts – Simulation-only Bitstream and Simulation-only Layer (GONG; DIESSEL, 2014).

emulate the physical fabric of FPGAs. The goal is to assist designers during activities involving the user design's testing/debugging/verifying in terms of reconfiguration components. Fig. 20 shows the main simulation artifacts for a ReSim simulation. In the simulations, simulation-only bitstream models two features of DPR. First, it exercises the transferring datapath. Second, it checks if the correct bitstream arrives in the Configuration Port (CP). About the CP artifact, it models the configuration port interacting with the user-defined reconfiguration controller. Regarding the Reconfigurable Region (RR), it is a placeholder of the Reconfigurable HW Modules that models DPR features such as Module Swapping and Triggering Condition, besides Spurious Outputs and Undefined Initial State. Thereby, ReSim only models reconfiguration aspects intending to find bugs in Dynamically Reconfigurable Systems designs, not providing or simulating a system-level view.

PReProS (BRITO et al., 2007a) is a general-purpose partially reconfigurable processor simulator based on a modified version of the SystemC kernel (BRITO; MELCHER; ROSAS, 2006) that allows to dynamically switch (activate/deactivate) modules during simulation – a DPR related feature. By modifying the processor parameters, the designers can evaluate its configuration, in a behavioral way, and the reconfiguration scheduler. When using the PReProS simulator (Fig. 21 (A)), the designer should just set the parameters and implement its blocks to configure the applications exchanging data with PRePros. According to Fig. 21 (B) (BRITO et al., 2007b), an initial (TLM) SystemC simulation is carried out, followed by a conversion to an HDL model aiming at implementation and comparison. PReProS focuses on the reconfiguration aspects, not simulating the application's behavior. Also, it lacks in power modeling, only dealing with reconfiguration performance and (HW) area usage metrics related to the reconfig-

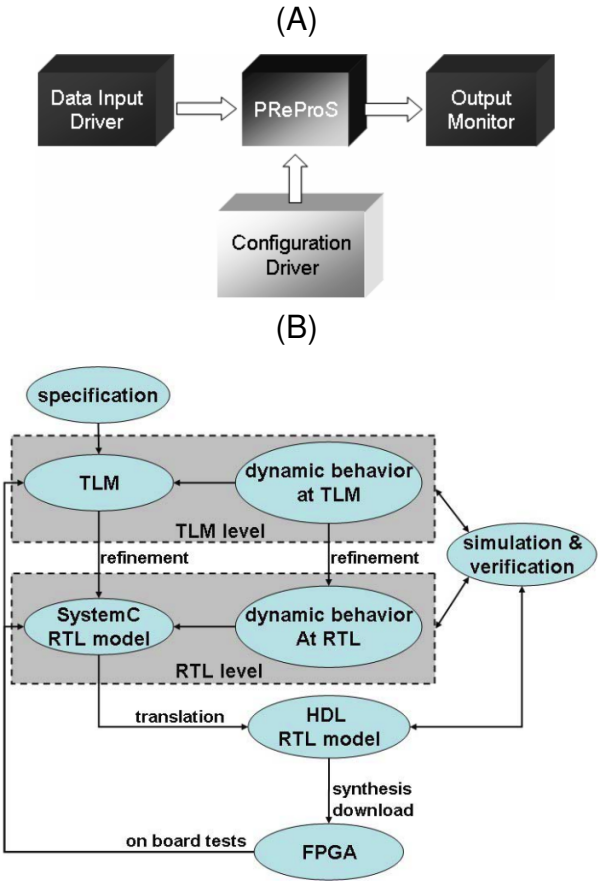


Figure 21 – (A) PRePros Overview (BRITO et al., 2007a) and (B) Adapted SystemC Design Flow (BRITO et al., 2007b)

uration process.

PAAS (Processor Accelerator Architecture Simulator) (LIANG et al., 2017) is built upon gem5 and Verilator, being gem5 responsible for simulating the CPU and memory infrastructure while Verilator is in charge of compiling Verilog code, generating an executable file that implements the FPGA's function. Fig. 22 shows the simulators' integration and the interactions between the system modules. Based on inter-process communication (IPC), the simulation occurs in parallel, aiming to reduce its time. Verilog describes the FPGA module implementation. PAAS deals with reconfiguration aspects by allowing user (manual) parameters specifying reconfiguration latency. The authors' principal focus is the interactions of an HW accelerator with the other system components, mainly memory sub-systems.

Another simulator based on the HDL simulator Verilator is HeteroSim (FENG et al., 2017). It also includes a modified version of Multi2Sim providing models for x86 cores, memory hierarchy, and cache coherence. HeteroSim co-simulates Verilog code and x86 executable, allowing a cycle-accurate simulation. Fig. 23 presents the HeteroSim Architecture, detailing the CPU part simulated by the modified Multi2Sim and the FPGA part. FPGA part includes components for kernel management and execution dealing

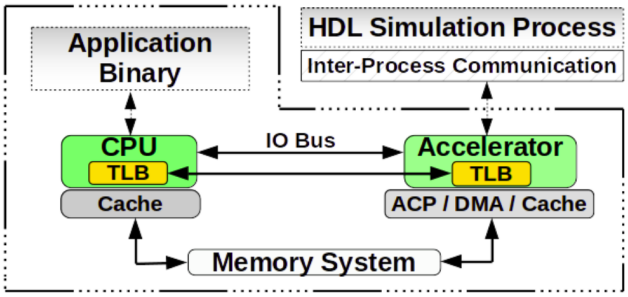


Figure 22 – PAAS Architecture (LIANG et al., 2017).

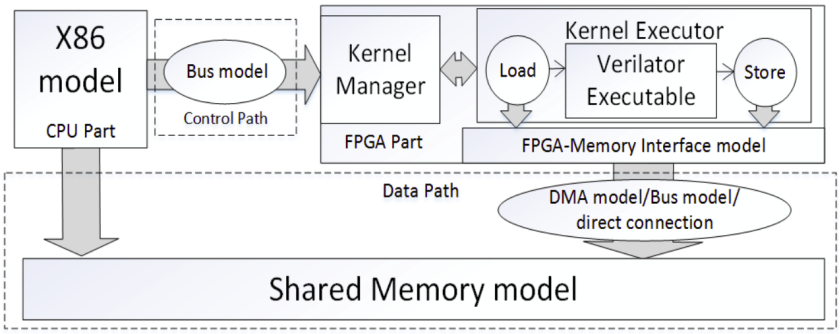


Figure 23 – HeteroSim Simulator Architecture (FENG et al., 2017).

with the definition of functions accelerated as kernels on FPGA, beyond a component that interfaces the FPGA memory with the shared system memory. Again, this work also puts more attention on the interactions with the memory system.

HeteroSim (FENG et al., 2017) and PAAS (LIANG et al., 2017) use a Verilator simulator to deal with FPGA tasks described in HDL (Verilog) code implementing DPR feature, but only PAAS deals with reconfiguration time. Thus, the simulators' utilization requires low-level code, a demanding artifact for initial design phases. Moreover, both works do not cover power/energy estimations during the simulations and do not employ multiple PRR.

QUADRI et al. (2010) present an application-driven design flow that employs high-level models using the UML MARTE profile in the modeling of reconfigurable SoCs. It uses an MDE (Model-Driven Engineering) approach through a model-driven co-design framework – Gaspard2. Based on UML MARTE models, the authors describe the control semantics for the system adaptivity. It focuses mainly on the dynamic reconfigurability of SoCs. The authors present two factors related to the modeling of Dynamic Reconfigurable Systems:

- (i) the description of the reconfigurable region that may have high-level application models specifying different and mutually exclusive implementations; and
- (ii) the modeling of the reconfigurable controller semantics responsible for managing the switching between the alternative modules related to the region.

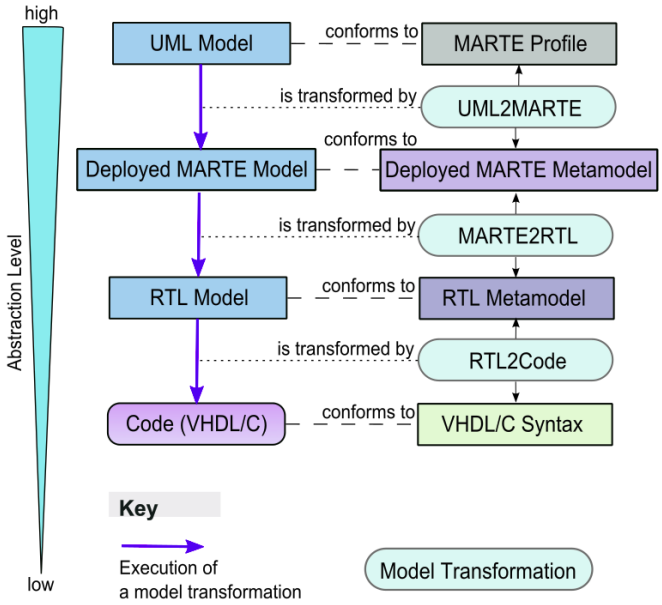


Figure 24 – The Model Transformation Chain in Gaspard2 (QUADRI et al., 2010)

A chain of model transformations enables the generation of implementation code from high-level models – according to Fig. 24. A model-to-text conversion translates the models of the factor (i) into an HW functionality (e.g., an HW accelerator and its different implementations subject to executing on an FPGA device). Regarding key aspect (ii), the control semantics provides the necessary elements (a state machine) to generate the source code related to the reconfiguration controller. As a case study, an anti-collision radar detection system validates the design flow. It uses the code generated from the model transformations to validate the modeled functionality through a *ModelSim* simulation. Also, synthesis with the *PlanAhed* design tool generates the appropriate files for the eventual implementation of DPR. Although the flow establishes high-level models employing DPR features targeting SoCs, it still applies standard tools – with time-consuming simulations/processing – based on low-level descriptions (HDL code) to simulate/evaluate/validate the design.

Centrifuge (HUANG et al., 2019) is a methodology and a flow (Fig. 25) that trusts in a HLS toolchain and an open-source simulator – FireSim (KARANDIKAR et al., 2019)) – to generate and evaluate heterogeneous SoCs. Aiming to accelerate the simulations, it uses an FPGA-enabled cloud platform (Amazon F1 instances) to simulate SW/HW stacks. The designer generates C code, possibly including a function (marked with *pragma* directive) to HW accelerate. This function code pass through a VIVADO synthesis producing an HW task implementation which is wrapped to allow access by the same interface. Modified LLVM compiler links the remaining C program through a RISC-V assembly code. The HW accelerator and the binary runs on a Centrifuge Accelerated SoC using the FPGA-based Cloud platform. Since the cloud instance accepts the generated files, the system can produce a complete SoC integrating the

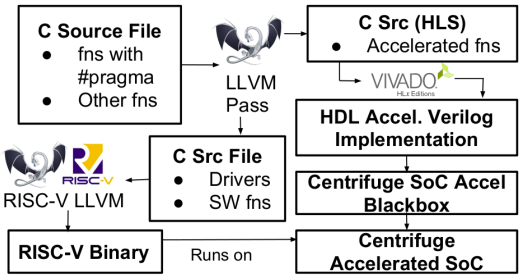


Figure 25 – Centrifuge HLS Flow and Tools (HUANG et al., 2019)

HLS-generated accelerator simulating it in a cycle-accurate way, running a complete SW stack. It supplies an end-to-end system evaluation capacity. The authors provide no data regarding the necessary time for HLS synthesis and system setup. Moreover, the simulation uses real devices (FPGAs) in its process demanding a ”large” FPGA cloud structure to allow the system setup and subsequent simulation. The Centrifuge uses standard design tools to generate RTL level code for simulation.

Lin-Analyzer (ZHONG et al., 2016) is a high-level accurate performance analysis tool for FPGA-based accelerators that enables rapid design space exploration considering various HLS directives (pragmas) without requiring RTL implementations. It leverages the dynamic evaluation by using dynamic data dependence graphs (DDDGs) generated from program traces that represent the dataflow of the accelerator under study. Making some assumptions regarding the resources of an FPGA device, Lyn-Analyzer makes the performance estimation of FPGA-based accelerators directly from high-level languages such as C/C++.

ZHAO et al. (2020) present COMBA, a comprehensive model-based analysis (COMBA) framework. COMBA uses analytical models based on design descriptions possibly containing directives (pragmas) related to functions, loops, and arrays, being capable of analyzing the effects of those pragmas based on an HLS toolchain. Fig. 26 presents the elements of the COMBA framework. First, a design description in C/C++ is translated to an LLVM IR, passing to the recursive data collector (RDC) that computes the parameters required by the analytical models based on the directives and pre-characterization information. The parameters fit in two categories: static information, e.g., the array element’s memory addresses, and dynamic information, e.g., the iteration latency of loops. The Performance Models deal with five frequently used directives: loop unrolling (LU), loop pipelining (LP), array partitioning, function pipelining (FP), and dataflow. On the other hand, the Resource Models estimate the required resources, namely, BRAMs, DSPs, and LUTs. Afterward, a two-stage metric-guided DSE (MGDSE-II) algorithm removes redundant design points (first stage) and uses evaluation metrics to identify performance bottlenecks, verify the resource constraints compliance, and define array partition type, in such a way indicating the directions of the exploration (second stage).

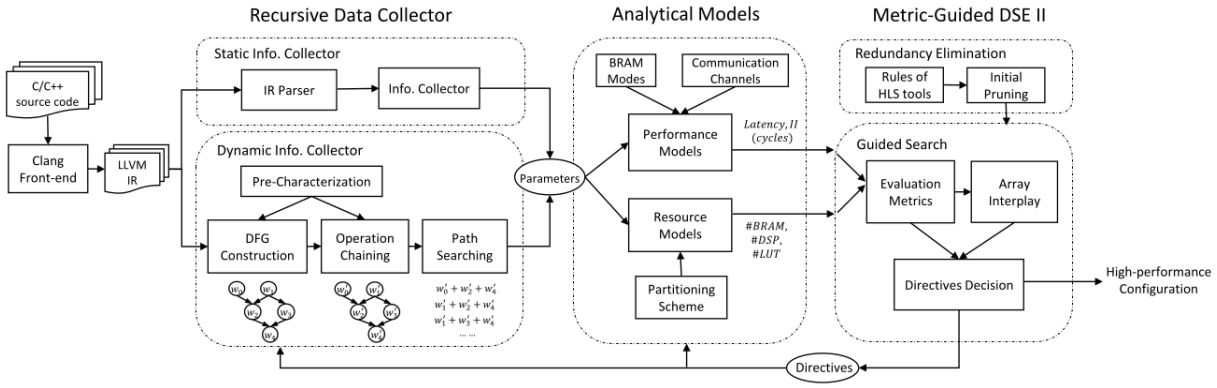


Figure 26 – COMBA Framework (ZHAO et al., 2020)

This works (ZHONG et al., 2016) (ZHAO et al., 2020) base its estimations on dynamic program traces using the LLVM IR to generate DDDG program representations. Thus, using characterization information regarding the HW resources of a device, and based on their analytical models, they produce estimations for FPGA-based accelerators. However, these estimations regard only the accelerator, not accounting for the whole system. Moreover, they only estimate performance and area metrics. Furthermore, features related to DPR and PRR are not explored.

3.2 A ”Bird’s Eye” View of the Related Works

Summarizing the presented works, Tab. 1 shows a comparative evaluation regarding aspects of heterogeneous systems containing reconfigurable components as HW accelerators during DSE activities. We define evaluation criteria that describe the previous works based on items related to simulation, platform description, reconfigurability, power estimation, performance estimations, and works main goals. We also include in Tab. 1 the aspects of the modeling and simulation infrastructure (including the FEHetSS simulator described in Sec. 4.4) developed and used in this thesis. The columns of Tab. 1 intends to map the previous aspects using the following evaluation criteria scheme:

- Simulation
  - What Simulation tool(s) is(are) used?
  - What Simulation Level of the HW Tasks?
- (Virtual) Platform
  - How is the Workload (Applications) description?
  - How is the Architecture description?
- Reconfigurability

Table 1 – Comparative Evaluation between the Related Works

	Simulation		*(Virtual) Platform*		*Reconfigurability*				*Power*	*Performance Metrics*			
Work	**Sim. Tool(s) ?**	**Sim. Level (HW Task) ?**	**WRKLD (Appl.) ?**	**ARCH ?**	**HW Accel. ?**	**DPR feat. ?**	**Reconf. Time ?**	**Sim. of Mult. PRR?**	**Power Model ?**	**Time ?**	**Power ? Energy ?**	**Utiliz. ? Area ?**	**Work's Main Goal?**
SAVE (MIELE et al., 2015)	SAVE (SystemC)	SystemC (TLM)	Tasks Graph (XML)	Generic Resources (XML)	Y †	N	N	N	Y (from HW Measur.)	Y	Y	Y	Resource Managem. Evaluation
gem5-Aladdin (SHAO et al., 2016)	Aladdin gem5	Pre-RTL	C to DDDG	Arch. Models	Y	N	N	N	Y (DDDG Charac.)	Y	Y	N	DSE, Interaction w/ Memory and other Arch. Comp.
HAPE (MAKNI et al., 2018)	HAPE	Pre-RTL & Analytical Model	C/C++ to DDDG	SoC Arch.	Y	N	N	N	Y (DDDG Charac.)	Y	Y	Y	FPGA Area & Power Estimation
Lin-Analyzer (ZHONG et al., 2016)	Lin-Analyzer LLVM	Pre-RTL & Analytical Model	C/C++ to DDDG	FPGA Device	Y	N	N	N	N	Y	N	N	FPGA Performance Estimation
COMBA (ZHAO et al., 2020)	COMBA LLVM	Pre-RTL & Analytical Model	C/C++ to DDDG	FPGA Device	Y	N	N	N	N	Y	N	Y	FPGA Performance &Area Estimation in DSE
rSesame (SIGDEL et al., 2009b)	rSesame	Discrete Event, trace-driven (KPN)	C/C++ to KPN	Reconf. Arch. (Molen)	Y	Y	Y	Y	N	Y	N	N	Task Mapping
FoRTReSS (DUHEM et al., 2015)	RecoSim	SystemC (CDFG)	CDFG	Arch. Models	Y	Y	Y	Y	N	Y	N	Y	PRR Design Eval. and Generation
(NOGUEIRA et al., 2016)	Parallel DEVS	Analytical Model	HSDF	ATG	Y	N	N	N	Y (Compon. Charac.)	Y	Y	Y	DSE of Multimedia Emb. Sys.
(AN; GAMATIÉ; RUTTEN, 2015)	CLASSY	CLASSY formalism - Abstract Clock Models	CLASSY - Tasks and its Sta./Dyn. Behaviors	CLASSY - Plat. PE, Platform Behaviours	Y	Y ‡	N	N	Y (Compon. Charac.)	Y	Y	N	High-level DSE of Adaptive MPSoC Appl.
(BLOUIN et al., 2011)	OSATE (AADL)	AADL Models	AADL Application Description	AADL Platform Description	Y	Y	N	N	Y (Compon. Charac.)	Y	Y	Y	Modeling FPGA on MPRSoC
PAAS (LIANG et al., 2017)	Verilator gem5	HDL (Verilog)	C to HDL	Arch. Models	Y	Y	Y	N	N	Y	N	N	Memory System Interaction
HeteroSim (FENG et al., 2017)	Verilator Multi2Sim	HDL (Verilog)	C to HDL	Arch. Models	Y	Y	N	N	N	Y	N	N	Memory System Interaction
Centrifuge (HUANG et al., 2019)	FireSim Vivado LLVM	RTL	C to Accel. HDL (RTL) (w/ Vivado)	FPGA Model + OS+Arch. Infrastruc.	Y	N	N	N	N	Y	N	Y	SoC Evaluation
MultiVers (LIGNATI et al., 2021)	Vivado	HLS with CPU-FPGA Cloud Envir.	HLS C	CPU/FPGA Cloud System	Y	Y	Y	Y	Y	Y	Y	Y	Cloud System Resource Provisioning
(HUANG; HSIUNG; SHEN, 2010)	Rhapsody (UML)	UML interacting with real HW	HW Task Template	Specific Device	Y	Y	Y	Y	Y	Y	Y	Y	HW/SW Co-design of Dynam. Partial. Reconf. Network Security Systems
COMPLEX (GRÜTTNER et al., 2013) (HERRERA et al., 2014)	SCOPE+ (UML)	SystemC	UML/ MARTE, Functional Code	UML/ MARTE, Platform Arch. Models	Y	N	N	N	Y (Compon. Charac.)	Y	Y	N	HW/SW Co-design
Gaspard2 (QUADRI et al., 2010)	Gaspard2	RTL (ModelSim)	UML/ MARTE (state graphs)	UML/ MARTE, FPGA Model	Y	Y	Y	N	N	Y	N	Y	FPGA-based SoC Design (DPR controller and PRR)
(GONG; DIESSEL, 2014)	ReSim	RTL	Simulition-only bitstreams	FPGA Model	Y	Y	N	Y	N	N	N	N	DPR Bug detection
(BRITO et al., 2007a) (BRITO et al., 2007b)	PReProS	(modified) SystemC, RTL	HDL	Reconf. Arch.	Y	Y	Y	Y	N	Y	N	Y	DPR Modeling & Design
This work	FEHetSS	SystemC (TLM)	OpenCL to Tasks Graph (XML)	Generic Resources (XML)	Y	Y	Y	Y	Y (PE Charac.)	Y	Y	Y	Early DSE for FPGA-enab. Heter. Systems

† *Even the authors have inserted HW Accelerator in the SAVE PE alternatives, its is not available in the respective implementation.*
‡ *Considering the CLASSY's alternative static behaviors of a task as an reconfiguration in a DPR System.*

  - Does the work employ HW Accelerators in its architecture?
  - Does the work employ DPR in its system assessments?
  - Does the work account PRR Reconfiguration Time in its system evaluations?
  - Does the work allow the evaluation of Multiple PRR in its Reconfigurable Regions?

- Power
  - Does the work provide a Power Model for its Reconfigurable HW Accelerators?

- Performance Metrics
  - Does the work provide Time or Latency Metrics?
  - Does the work provide Power/Energy Metrics?
  - Does the work provide Utilization/Area Metrics?

Considering the related works' characteristics presented in Tab. 1, Tab. 2 shows the positioning of each described initiative based on the features Simulation Level (high or low), Power Model (Yes or No), DPR (Yes or No), and PRR (Yes or No). This positioning highlights the context of this Thesis in the area.

Based on Tab. 1 and Tab. 2, we can observe some gaps in the area, mainly regarding the simulation of (detailed) low-level models. Low-level simulations are impractical in the early stages of the system's design cause of its long simulation time. Besides the absence of a power model – unable to generate power/energy estimates for theirs HW accelerators. Moreover, although the works consider HW accelerators in their system's architecture, DPR and PRR features are not fully utilized, sometimes not accounting details such as reconfiguration time.

The methodology and simulator FEHetSS, both developed in this thesis, in contrast to the related works, allows us to model a Heterogeneous System Architecture (HSA) including *VirtualHWAccel* (FPGA), simulating its behavior using SystemC modules in TLM abstraction level. Moreover, FEHetSS is a system-level simulator that enables modeling FPGA-Enabled architectures at a high abstraction level, modeling the architecture as a pool of resources and the applications as a tasks graph. In this work, we use OpenCL concepts in our models for both applications and processing elements. A virtual platform (VP) represents the input in FEHetSS. This structure describes the architecture (processing elements - PE) and the workload (applications and its arrival time), beyond the mapping of the applications' tasks (kernel and host modules, corresponding to "elaboration" and "main" tasks, respectively) to the available PE. Furthermore, it provides estimations – even in early design phases – for time, power/energy,

Table 2 – Related Works Positioning based on Simulation Level, Power Model, DPR, and PRR Features

		High-Level Simulation		Low-Level Simulation	
		PRR	**No PRR**	**PRR**	**No PRR**
Power Model	**DPR**	*This Work FEHetSS (Mod.&Sim. Infras.)*	*CLASSY (Abs. Clock Mod.) (AN; GAMATIÉ; RUTTEN, 2015)* *AADL (BLOUIN et al., 2011)*	*UCoP (DPRNSS) (HUANG; HSIUNG; SHEN, 2010)* *MultiVers (LIGNATI et al., 2021)*	
	No DPR		*SAVE (MIELE et al., 2015)* *DEVS (HSDF+ATG) (NOGUEIRA et al., 2016)* *COMPLEX (GRÜTTNER et al., 2013) (HERRERA et al., 2014)*		*gem5-Aladdin (SHAO et al., 2016)* *HAPE (MAKNI et al., 2018)*
No Power Model	**DPR**	*FoRTReSS (DUHEM et al., 2015)* *rSesame (SIGDEL et al., 2009b)*		*ReSim (GONG; DIESSEL, 2014)* *PReProS (BRITO et al., 2007a) (BRITO et al., 2007b)*	*PAAS (LIANG et al., 2017)* *HeteroSim (FENG et al., 2017)* *Gaspard2 (QUADRI et al., 2010)*
	No DPR				*Centrifuge (HUANG et al., 2019)* *Lin-Analyzer (ZHONG et al., 2016)* *COMBA (ZHAO et al., 2020)*

and utilization, also regarding FPGA features like the employment of multiple PRR and the use of DPR accounting for its reconfiguration latency and power consumption.

3.3 Research Challenges

Evaluating the related work, we identify gaps that can be filled up by this thesis. Most works deal with FPGA simulation but using low-level representations, as HDL code (in RTL level), to describe the circuit implemented in the HW device. RTL simulation is a high time-demanding task that specialized tools and designers must conduct, maybe impractical in early design phases. Coarse-grained models in a high-abstraction level are more easily defined/simulated. Moreover, some works provide DPR features, but again based on low-level descriptions, or even being the only simulated aspect, not estimating the workload execution itself in a system-level view. Further, even regarding DPR features, some works do not account for reconfiguration time. Besides, the modeling of multiple PRR is a rare situation.

Another aspect is the absence of a power model in several works. Power/Energy related estimations are essential metrics to subsidize the decisions during the system design. Together with execution time, energy/power metrics make the design points

trade-off analysis richer. Also, utilization estimates (in terms of percentage of time or consumed HW area) provides greater precision to the profiled solution, adding an extra dimension for evaluation.

Considering the identified gaps, and before presents the research challenges of this thesis, we recall our **motivational scenario**: *Consider a design initiative for a new embedded system. It must deal with a specific workload and considers metrics such as execution time, power/energy consumption, and processing elements/HW area utilization. Since probably no physical architecture is available, early design evaluations must happen using high-level models. Thus, System-Level Simulation is a way to assess the design points employing Virtual Platforms (VPs) modeling the system architecture, the workload, and their mappings. With notions such as host/device processors and application kernels, OpenCL concepts serve as a basis for modeling the system architecture&workload. Also, considering High-Level Synthesis tools, it outcomes appropriate artifacts that enable modeling the application's kernel(s) running in FPGA units. FPGAs are customizable devices usually providing performance and power efficiency. Besides, it offers features like Partially Reconfigurable Regions (PRR) and Dynamic and Partial Reconfiguration (DPR). Each PRR can potentially act as an independent HW accelerator. DPR enables dynamically change the PRR's configuration. Moreover, the custom capability of FPGA devices allows experimenting with diverse HW task implementations aiming to obtain a good trade-off between execution time, energy consumption, and resource utilization. Given this scenario, how could a designer conduct early design space exploration (early-DSE) activities in a feasible time considering high-abstraction descriptions of the system and its applications, producing valuable design artifacts for initial decisions making? All of this, before passing to more detailed and low-level HW/SW development.*

Now, we can describe the research challenges of this thesis, as follows:

- **Context**: Regarding a System-level Simulation infrastructure enabled to (i) include HW tasks mapped in reconfigurable HW Accelerators (as an FPGA); employing (ii) high-abstraction level models in early design phases; (iii) simulating the models at the same level; and (iv) supporting system estimations for time, energy/power, and utilization.
  - **Challenge 01 (C01)**: *How to enable early-DSE on heterogeneous systems that include reconfigurable hardware acceleration through FPGAs?*
- **Context**: When employing an FPGA device as an HW accelerator, multiple implementations of an application kernel (as an HW task) are possible (e.g., a solution using several compute units to accelerate the execution of the many items to process or other that makes loop unrolling to diminish the iterations). Considering models of a Virtual Hardware Accelerator (*VirtualHWAccel*) and other

components of a virtual platform (VP), a designer must be capable of evaluating the application (and its kernel) using metrics like execution time, power/energy consumption, and HW utilization.

  - **Challenge 02 (C02)**: *How to assess (at early stages) different architectural FPGA implementations for a specific application kernel, as an HW task, within a heterogeneous system?*

- **Context**: Considering: (i) the characteristics of the Partially Reconfigurable Modules (PRM) (application's kernels implemented as HW tasks) to be configured; (ii) the dynamic reconfiguration of the PRR while simulation goes on; and (iii) the related reconfiguration time and its consumed power.

  - **Challenge 03 (C03)**: *How to model FPGA units at a high level, supporting PRR and DPR, focusing on high-level system simulation?*

Based on the listed Research Challenges, this thesis proposes a modeling and simulation infrastructure to perform early Design Space Exploration (DSE) for heterogeneous systems featuring FPGA devices concerning the reconfigurable hardware acceleration, including aspects related to the architectural implementation of HW modules and considering partial reconfiguration features such as the device's partition in PRR and the DPR flow. We discuss the infrastructure's modeling aspects in Chapter 4 – Sec. 4.3, followed by the simulation's detailing in Sec. 4.4. Moreover, we planned Case Studies performing experiments for the infrastructure evaluation (in Chacontainingpters 5 and 6).

3.4 Chapter Summary

In this chapter, we presented some works that deal with HW accelerated systems' modeling/simulation. These systems usually employ FPGAs as a *VirtualHWAccel*s. Some research initiatives take advantage of the FPGA's flexibility utilizing reconfiguration features. But, few regard the reconfiguration time and experiment a different number of PRR. The simulation level employed in the works frequently resorts to low-level models, likely impacting the simulation time. A power model and the generation of power/energy estimations do not prevail in the related academic literature. The use of HW devices itself appears as a design method alternative. But, it possibly restricts the works' applicability or maybe demands major HW infrastructures.

Moreover, based on identified gaps in the area, we list some research challenges to tackle in this thesis. In essence, the challenges involve the modeling aspects related to the VP and the use of HW tasks implemented in FPGA devices as a *VirtualHWAccel*, beyond the definition of the number of PRR and the employment of DPR. This

modeling shall consider high-level models for architecture and workload, aiming to employ in early DSE. The system's simulation shall occur at an appropriate level. Also, allowing rapidly-produced output evaluations containing estimations for Power/Energy, Time, and Utilization. So, an early power model shall be available. Furthermore, the challenges point out the necessity to deal with DSE initiatives and experimentation with different architectural FPGA implementations of an application's kernel.

4 FPGA-ENABLED HETEROGENEOUS SYSTEM MODELING AND SIMULATION

This chapter presents the core of the thesis describing its main contributions. The focus of it is on FPGA-enabled Heterogeneous System Modeling and Simulation aiming at Early Design Space Exploration (DSE). Virtual Platforms (VPs) models describe these systems allowing to use of Virtual HW Accelerators (*VirtualHWAccel*)s – modeled as FPGA devices. Partially Reconfigurable Regions (PRRs) partition the *VirtualHWAccel* permitting to deal with each PRR as a potential HW accelerator. Dynamic and Partial Reconfiguration (DPR) enables the system to modify PRRs using Partial Reconfigurable Modules (PRMs) according to the workload's demand.

Fig. 27 presents the conceptual elements embraced by this thesis. The yellow boxes indicate the chapter/section that presents the associated element. We envision a scenario where a designer can model the system as a Virtual Platform (VP), simulating its behavior considering a workload, and producing appropriate metrics to use in a Design Space Exploration (DSE). Two model(s) sets compose a VP. An Architecture model describes the Processing Elements (PEs) integrating the system. And a Workload model describing the applications to submit to the system. A designer produces those models by employing methodological steps, tools, and artifacts defined in the *Architecture and Workload Modeling* methodology. The produced models are organized in a VP repository, being available to direct use as the input of a Simulation Infrastructure (FEHetSS) or via a DSE environment driven by an Optimization Heuristic. FEHetSS simulator uses the input models, especially their annotations, to derive log metrics regarding time, power, energy, and utilization.

Considering the Architecture model, Fig. 27 presents a VP containing five PEs – three CPUs, one GPU, and one FPGA. A PE model includes a power model indicating its operation frequency, power, and idle power. In the case of an FPGA, the modeling must regard some specific aspects. First, we can partition the device into PRRs, each one capable of implementing an HW task. Besides, the device has a Periphery HW responsible for communication and input/output (I/O) – e.g., transceivers and GPIO (General-Purpose I/O). And a board interface (BI), a static region that facilitates com-

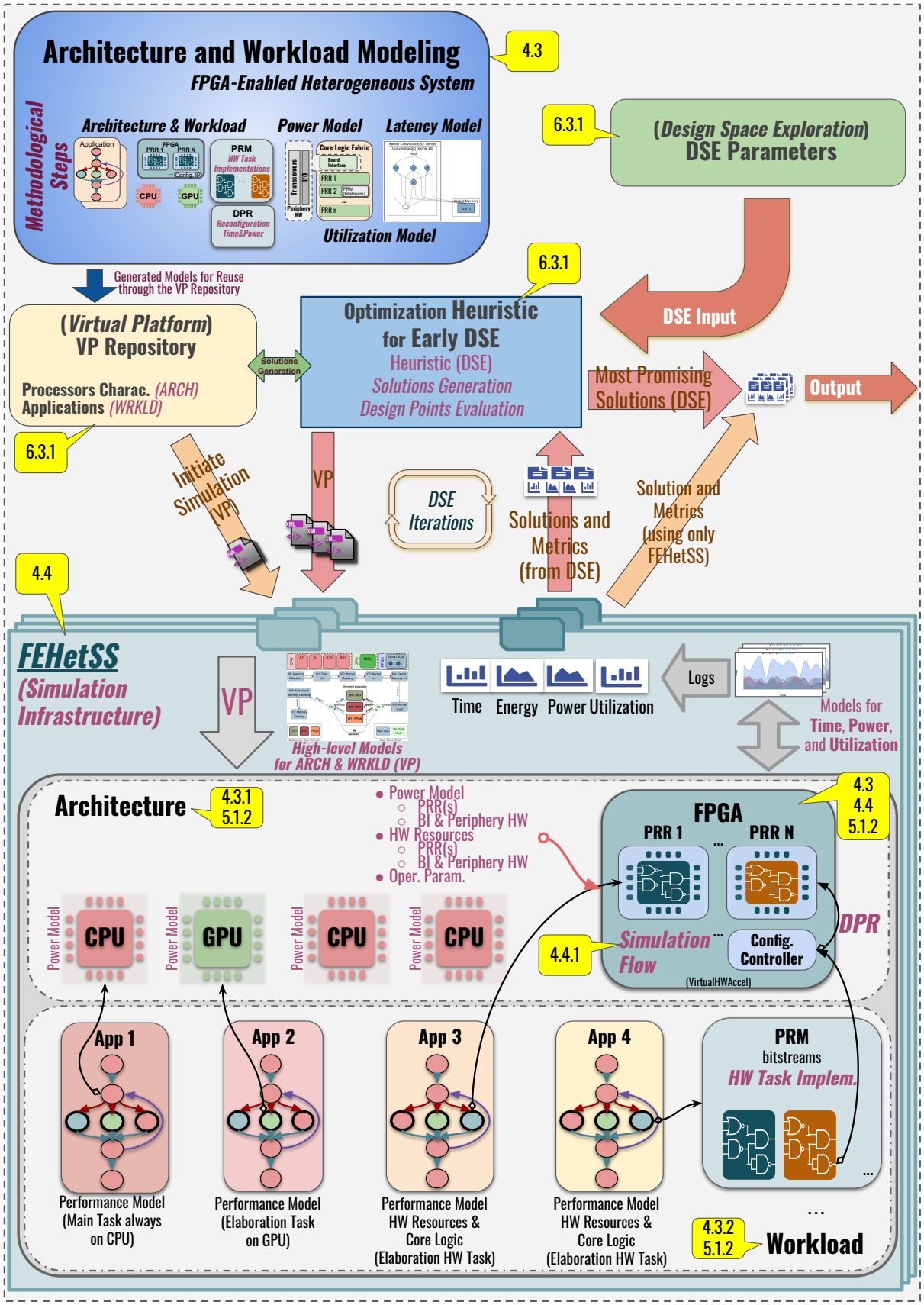


Figure 27 – Conceptual Elements and Main Contributions of the Thesis.

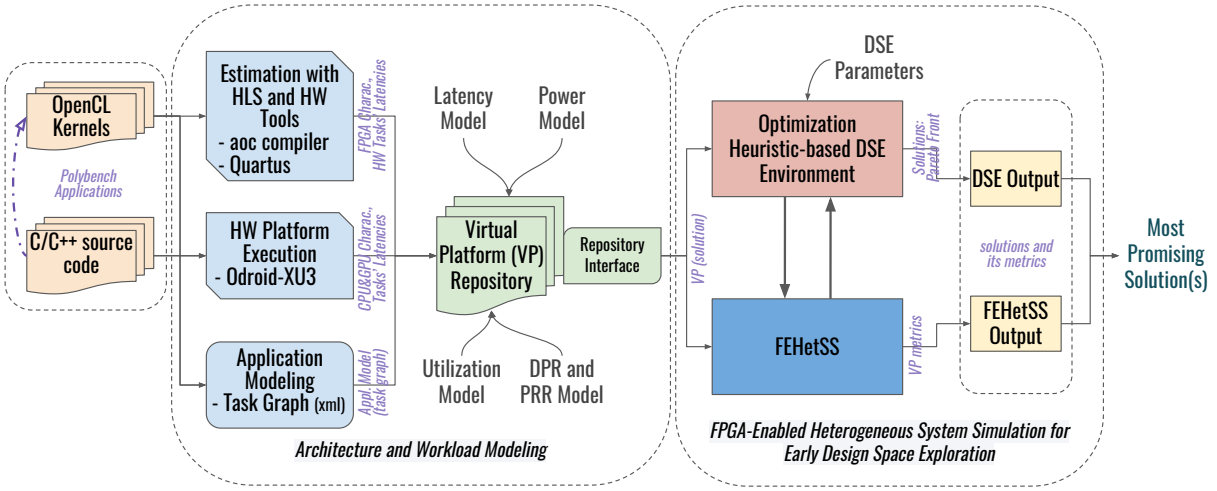


Figure 28 – FEHetSS Framework's Flow

munication with external interfaces. Each one has its power characterizations. Second, the reconfigurable regions (PRRs) can pass through a Dynamic and Partial Reconfiguration (DPR). It configures a PRR consuming time and power, using a bitstream passing by a configuration controller. It configures one PRR at a time but maintains the remaining PRRs in operation. Last, a simulation flow drives the FPGA mode (state) transitions, managing the simulation in the device and PRRs scopes.

About the Workload model, Fig. 27 shows four applications in execution as the system's workload. Tasks graphs describe the application's structure indicating the tasks and their types. CPUs sequentially executes *main* tasks. Regarding the *elaboration* tasks, all PE types (CPUs, GPUs, and FPGAs) can execute them in parallel. FPGA device provides the capability of executing HW tasks in a custom and optimized way. Main tasks manage the execution and prepare the Elaboration tasks allocation. The latter represents the application's kernels. The graph's edges indicate the execution flow via its forward (normal advance), backward (for loop structure), and branch (triggering the elaboration tasks) variety. Regarding elaboration tasks executing in FPGA, an HW task implementation (represented by a bitstream) configures a PRR preparing its execution as a Virtual HW Accelerators (*VirtualHWAccel*)s.

Fig. 28 shows the flow describing the steps during the creation of the models in the VP repository and its use through the DSE environment or directly via FEHetSS. Considering the DSE environment, it uses the VP repository to create any necessary VP. A set of parameters defines the range and possibilities for the solutions' components. An Optimization Heuristic directs the DSE using FEHetSS as a Design Point performance evaluator. FEHetSS provides metrics for each simulated VP. Through DSE iterations, the heuristic produces an outcome with the most promising solutions.

In summary, the *main goal of the work is to provide a Modeling and Simulation infrastructure. It aims to support Early Design Space Exploration (DSE) for heteroge-*

neous systems featuring FPGA fabric for reconfigurable hardware acceleration. Moreover, the proposed infrastructure also enables HW architecture design space exploration regarding the HW/SW interface into a heterogeneous system.

We structure the remaining of this chapter in the following way. In Sec. 4.1, we present a methodological view of the work. After, in Sec. 4.2, we describe Task-level Modeling aspects passing through the idea of Heterogeneous Task-level Parallelism (HTLP) and the Simulator SAVE-htlp (BETEMPS et al., 2018) developed during this work aiming to deal with HTLP. Sec. 4.3 presents the methodological steps employed in VP modeling, mainly its Architecture and Workload. The main aspects of the methodology are:

- The architectural organization of the Processing Elements, including *Virtual-HWAccel*s modeled as FPGA devices, and enabled to organize the device in multiple PRRs;
- The DPR simulation regarding the Reconfiguration Time of a PRR and the whole device;
- The Power Model for the FPGA device and their PRR, characterizing the Power Consumption during Simulations, even during a PRR/device (re)configuration;
- The applications' modeling, regarding its structure and tasks' latency, including the latency of the kernel when implemented as HW tasks executing into an FPGA's PRR;
- The Tools and Steps to estimate latency and power of the application's tasks, including the execution of kernel(s) in CPUs, GPUs, and especially in FPGA devices; and
- The steps to take measures in a real HW Platform. These measures allow annotations in the models describing the Processing Elements (PEs) and Applications.

Lastly, we introduce the simulator (FEHetSS) built during the development of the thesis (Sec. 4.4). FEHetSS regards the described models aiming to simulate VPs. We describe the FEHetSS main components, its input structure, and their mode-based simulation flow.

4.1 Methodological View of the Thesis

The VP modeling methodology and the FEHetSS simulator make up an HW&SW system design framework aiming at the evaluation of candidate solutions in Early DSE circumstances. From a methodological angle, Fig. 29 describes a broad view of the thesis. As researchers during the development of the work, we actuate in several branches, as following discussed.

OpenCL Model. We adopt the concepts of OpenCL to model the workload applications, especially the concepts involving OpenCL's Platform and Execution models.

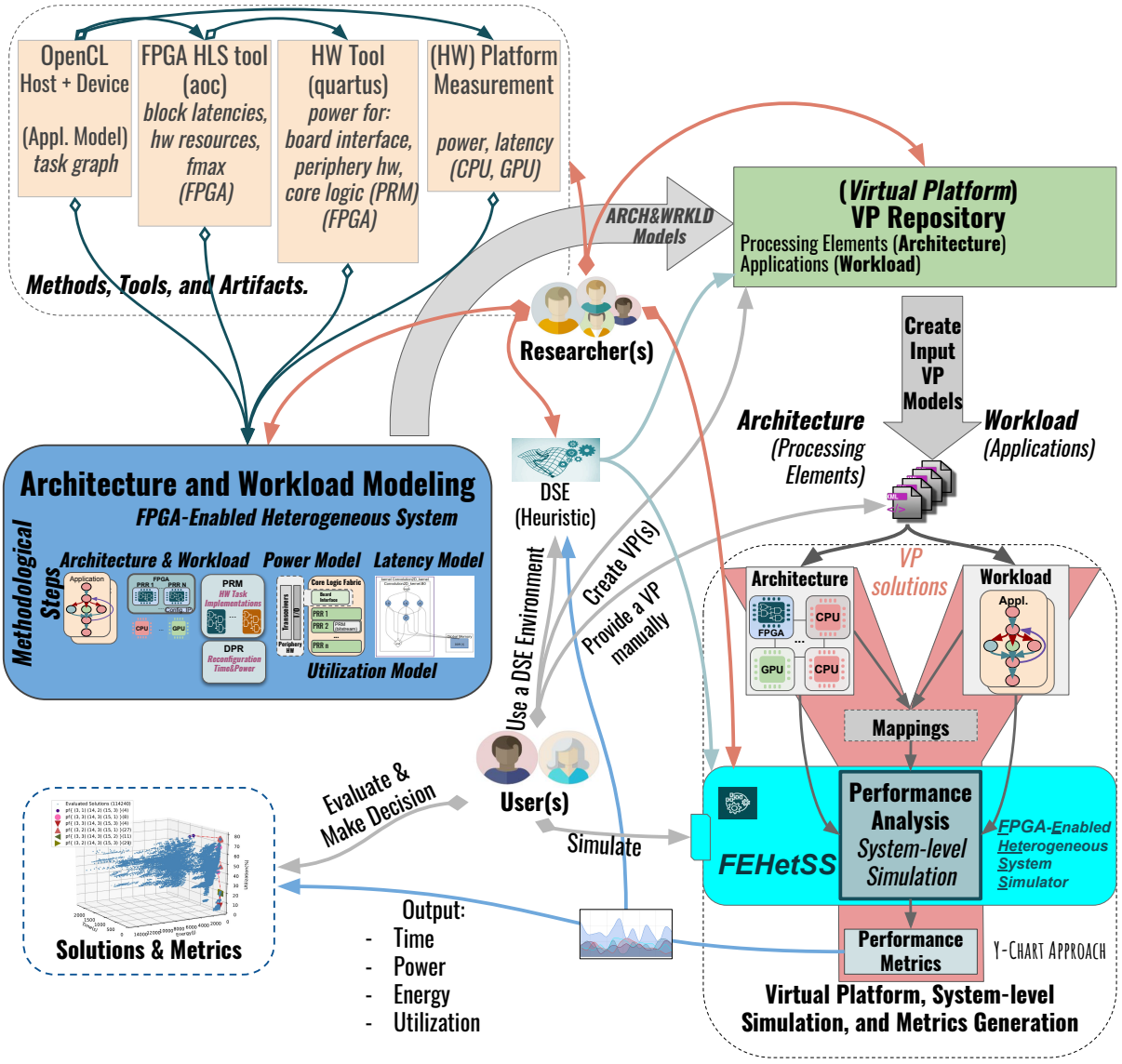


Figure 29 – Methodological View of the Thesis.

During an application modeling, we abstract the existence of tasks that execute on a Host and those allocated on a Device. These tasks correspond to the *main* and *elaboration* tasks previously described, respectively. In a platform, usually, a CPU acts as a Host. On the other hand, CPUs, GPUs, FPGAs, DSPs, or other processing elements can act as a Device. Moreover, we employ an OpenCL benchmark to serve as applications' examples, modeling them based on these conceptions.

HLS Tool. Regarding the modeling aspects of FPGA devices, we employ a High-Level Synthesis (HLS) tool (`aoc` compiler) capable of providing the kernel and device-related estimations – power and latency. HLS tool outcomes several reports and HW design files. It allows extract kernels' latency, HW resources, and maximum operating frequency. Besides, we use a power estimation tool (quartus/PowerPlay) based on modified HW design files (from the HLS synthesis process) to estimate the power consumption of an HW task module (core logic) in a PRR and other static regions of a

device.

HW Platform Measurement. Since we aim to model heterogeneous systems, CPU and GPU model characterizations are necessary. For these PEs, we depend on a real HW platform measurement employing stamp statements in OpenCL code. We use a time counter to extract latencies. For power-related extractions, we apply the data provided by platform-embedded energy sensors. We employ the Odroid-XU3 (HARDKERNEL, 2020) board as the real HW platform.

Architecture and Workload Modeling. We define a methodology to derive VP models. We base our modeling method on the Y-Chart approach, defining independent modeling of architecture and workload. A mapping statement indicates what processing element to allocate for an application task. As described earlier, CPU is the target mapping of *main* tasks. On the other way, an elaboration task can map their threads on CPUs, GPUs, or FPGAs. The methodology also includes aspects related to the following elements:

- *Steps to estimate the Power of*:
  - A PRM (HW task module) – a synthesis with the `aoc` compiler (INTEL, 2020a) characterizes a PRM implementation by generating several design files and reports. Based on those files, we employ the PowerPlay tool (integrated into Quartus (INTEL, 2020b)) estimating power consumption for an HW task module;
  - A PRR in a partitioned FPGA device – initially, in a simulation, a blank bitstream configures a PRR. It allows power saving until a DPR occurs configuring the PRR with a given PRM's bitstream. After that, the PRR passes to consume the respective power of the just configured PRM. This process repeats at every reconfiguration;
  - The reconfiguration's method during a DPR – During the DPR, we model the reconfiguration time according to the PR controller bandwidth and the bitstream size of the incoming PRM. For the power consumption through the DPR, we employ the Medium Grained Model presented in (BONAMY et al., 2014) (represented by Eq. 2). It produces an interpolation of the static power before DPR and that one immediately after; and
  - The static regions of an FPGA device – Quartus tool (INTEL, 2020b) also provides estimations for the static regions of a device. We use these estimations to model the power consumption in such regions.
- *Equations to estimate the kernel/block latencies while executing into an FPGA's PRR* – the `aoc` compiler tool, integrated into Intel FPGA SDK for OpenCL Software (INTEL, 2020a), provides reports estimating the latency of the kernels' blocks. We model the latencies by applying a set of equations (3 to 6);

- *Annotation of performance models regarding the tasks* – in an application description, a performance model describes each task including the reference frequency and its latency. For elaboration tasks, there may be performance models for each possible type of processing element. In the models, we insert annotations based on real HW platform measurements or even estimations produced by HLS tools (in our case, the `aoc` compiler) – the latter for the HW tasks implemented in FPGA devices;
- *Annotation of power models of CPU and GPU* – for each processing element a power model is profiling it including an operating frequency and the respective power and idle power. For CPU and GPU, measurements on a real HW platform provide the subsidies to generate the model's annotations;
- *Creation of a task graph describing the applications* – considering the common functions' structure of an OpenCL program, we define a general task graph capable of describes an application. Considering real HW platform measurements or tool estimations, we profile each application's task;
- *Annotation of an FPGA device model describing their power model, HW resources, and DPR parameters* – an FPGA device model contains some additional details comparing with other PE types. For power, we employ estimations from Quartus generated from a kernel considering a specific device. Regarding the HW resources, the `aoc` compiler tool (INTEL, 2020a) provides reports detailing the necessary HW for each FPGA portion (dynamic and static regions). The DPR parameters include data about the bitstream size (from `aoc` generated configuration file) and the reconfiguration controller bandwidth (based on the Intel PR controller IP (INTEL, 2020b)); and
- *Annotation of an HW task indicating their HW resources, power consumption, and bitstream size* – for HW tasks, mapped to an FPGA unit, we annotate it with data about the necessary HW resources to FPGA-implementing the task, and its core logic information (composed by a frequency and its dynamic and static powers). The `aoc` compiler tool (INTEL, 2020a) provides the HW estimation and the bitstream size while Quartus estimates the powers.

FEHetSS simulator. We implement our model conceptions in a System-Level Simulator. FEHetSS (FPGA-Enabled Heterogeneous System Simulator) is responsible for the performance analysis in our approach, simulating VPs and logging their metrics. FEHetSS is written in SystemC using TLM as the abstraction level. It allows the embedding of a Virtual HW Accelerator *VirtualHWAccel* – modeled as an FPGA device – in system architecture. The *VirtualHWAccel* includes features of DPR and PRR.

VP repository. We prepare a Virtual Platform (VP) Repository. It contains the model's portions that are combined to form a VP. We also define automation scripts that receive a VP description producing the appropriate VP model aiming to simulate it

in FEHetSS.

DSE Environment. We envision our Modeling&Simulation infrastructure supporting Early DSE activities. Such an infrastructure frequently requires some DSE's support. Usually, the design space is vast and exhaustive exploration is unfeasible. Optimization Heuristics can prune the design space and yet still supplying quality solutions – with "quality" meaning solutions that are close to the optimal ones regarding a set of objective functions.

Considering Use Cases for FEHetSS Users, she/he can use the VP repository automation scripts to create solutions providing a VP description or even creating a whole solution manually. Based on an existing candidate solution, users can initiate simulations with FEHetSS and evaluate the output metrics. In a more scalable scenario, a user can employ a heuristic-based DSE environment. FEHetSS supports such integration allowing that a heuristic-driven DSE initiates several simultaneous simulations evaluating candidate solutions through a series of iterations.

4.2 Task-level Modeling and Heterogeneous Task-level Parallelism

As described in Chap. 3, we have accessed the SAVE simulator (MIELE et al., 2015) source-code. This simulator's code is the base for the implementations developed during this thesis. First, we identify that SAVE does not provide the capability of splitting same purpose tasks in parallel threads, mapping them on different PEs types. Hence, the original framework was modified, and we named this extended version as SAVE-htlp (BETEMPS et al., 2018).

4.2.1 The SAVE-htlp Simulator

We develop some extensions in the original framework (MIELE et al., 2015) to deal with Heterogeneous Task-Level Parallelism (HTLP), making up the SAVE-htlp simulator (BETEMPS et al., 2018). In SAVE-htlp, a task graph describes an Application containing Nodes and Dependencies sets. A Node represents tasks of two types:

- (*i*) Main tasks: which are application portions always executed sequentially by the CPUs;
- (*ii*) Elaboration tasks: representing kernels executed by CPUs, GPUs, FPGAs, or other PEs – likely in a parallelized and heterogeneous way.

Nodes contain a Performance Model consisting of latency and frequency values. A Dependency represents edges of three types:

- (*i*) forward: indicates the application's advance;

- (*ii*) backward: represents a loop edge; and
- (*iii*) branch: indicates the advance to an elaboration task (a kernel). Distinct PEs can execute it.

SAVE-htlp allows to start concomitant threads of different elaboration tasks implementations – i.e., implemented for distinct PEs – although all tasks/threads being part of the same application, only indicating the possible executing models (e.g., CPU, GPU, and FPGA). It characterizes the employment of heterogeneous parallelism. In this context, there is a forward movement to a task set using branch edges. This set triggers its threads in distinct PEs. It allows parallelizing an application's kernel (elaboration task) in a heterogeneous way. E.g., running in CPU and GPU units in parallel. Moreover, SAVE-htlp has a controller to handle the join (in forwarding tasks) of the created threads. The original framework allows threads created from only one elaboration task per time and application. The number of created threads from an elaboration task set (in a TLP portion) can be explicitly defined and different for each TLP application portion. Parameters in the application graph model allow these aspects. SAVE-htlp is capable of dealing with nested loops by resetting the cycle counters. It aims to deal with more complex application models that can contain many nested loop structures.

To illustrate the HTLP concept, Fig. 30 shows models for three sample applications and the Odroid-XU3 architecture (HARDKERNEL, 2020). On the upright of Fig. 30, two application model samples employ homogeneous parallelism using eight threads in CPU and GPU, respectively. On the up left, the model uses heterogeneous parallelism via the creation of eight threads to be executed concurrently in CPU and GPU. The attributes *hetParall* and *maxThreads* (Fig. 31) defines each case.

SAVE-htlp receives an input describing the virtual platform (VP) as a XML file. Fig. 31 conceptually describes the input elements relationship by using an UML class diagram (GOMAA, 2011). Parameters *hetParall* and *maxThreads* define the use (or not) of heterogeneous parallelism and the number of employed threads, respectively. The DEPENDENCY class (an edge in a task graph) of Fig. 31 and the annotations in Fig. 30 define these parameters.

Regarding the SAVE-htlp experimentation, we define the following main goal (BETEMPS et al., 2018): "*to evaluate the HTLP by exploring the different kernels characteristics of the BMA case study application by the use of system-level simulation*". Next, we present the SAVE-htlp case study.

4.2.2 SAVE-htlp Case Study

We use a BMA (Block-Matching Algorithm) application as a case study for SAVE-htlp, using an implementation presented in (MELO et al., 2016). Video coding (e.g., Motion Estimation) and computer vision (e.g., Object Tracking and Stereo Matching)

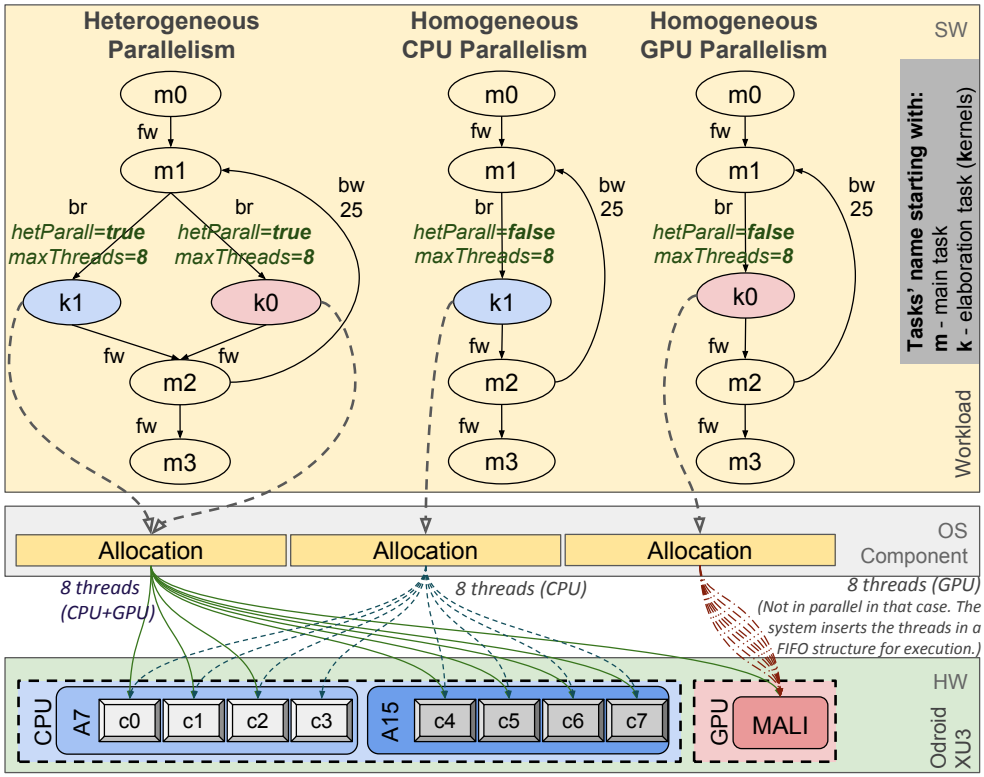


Figure 30 – Heterogeneous Parallelism vs Homogeneous Parallelism.

domains normally use BMA applications/modules. Besides, the Odroid-XU3 platform (HARDKERNEL, 2020) is the base architecture model. Fig. 32 shows the BMA application model used in the SAVE-htlp experimentation (BETEMPS et al., 2018).

BMA applications must use similarity criteria such as, e.g., the Sum of Absolute Differences (SAD). The original BMA implementation (MELO et al., 2016) uses SAD aiming at GPU and FPGA processors. BMA has certain parallelization aspects making it suitable to be explored in HTLP. In the BMA model (Fig. 32), each node presents its mapping (CPU or GPU), latency, type, and input/output data. The tasks latencies incorporate the memory operations delays. Thus, we have used zero values for all task input and output data parameters. Thus, the model presents these values only for modeling purposes. Regarding HTLP, we can observe three points (elaboration tasks) able to parallelize the BMA application:

- *srcB_filling*: data preparation for *sad_execute* task. According to the number of threads initiated for *sad_execute*, an equal one is triggered for this task.
- *sad_execute*: execution of SAD calculation. The number of threads for this task changes the cycles in the edge (*t14*, *t05_block_X*) — denoted by N in Fig. 32. As more threads started, fewer iterations will be needed.
- *sad_grouping*: grouping of SAD values — from 8x8 pixels size blocks into 64x64 pixels size CTU blocks.

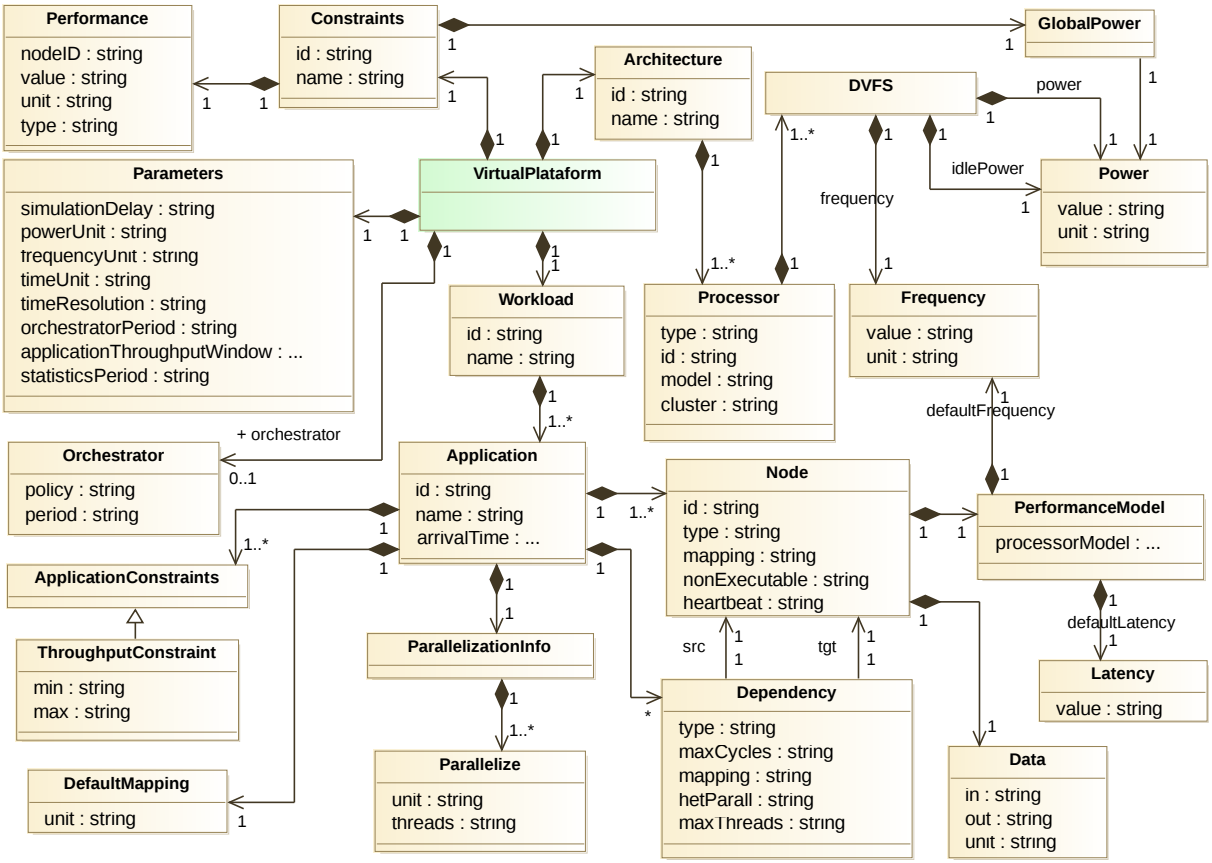


Figure 31 – SAVE-htlp VP Input Elements and their Relationships.

Tab. 3 presents the SAVE-htlp experiment settings, including the configuration *id* (#), the allocated PE type(s), the used number of threads, and the number of processors (column `A15`, `A7`, `Mali`) for each PE type available in the architecture for the *srcB_filling*, *sad_execute*, and *sad_grouping* tasks execution. Furthermore, we use three SAVE-htlp resource mapping policies:

- `Base`: allocates all available CPUs (A15 and A7 clusters, in such order), according to the number of threads;
- `A7`: allocates only the A7 CPUs, according to the number of threads;
- `A15`: allocates only the A15 CPUs, according to the number of threads.

In the configurations #1 to #4, only one thread is triggered for each elaboration task. The PE type used in the *srcB_filling* task is always CPU since it has a nature of a memory transfer task. In all the configurations, the task *sad_grouping* uses only one thread since it is a sequential procedure with abundant data parallelism. For the same task, the used PE type is CPU or GPU (exclusively) in each configuration. Exceptionally, setting #3 uses the CPU and GPU in an alternating way. The *sad_execute* task has configurations with only one type of PE (CPU or GPU) except for the settings #3 to #10. In #3 to #4 is used the alternated mapping between CPU and GPU. Settings #5 to #10

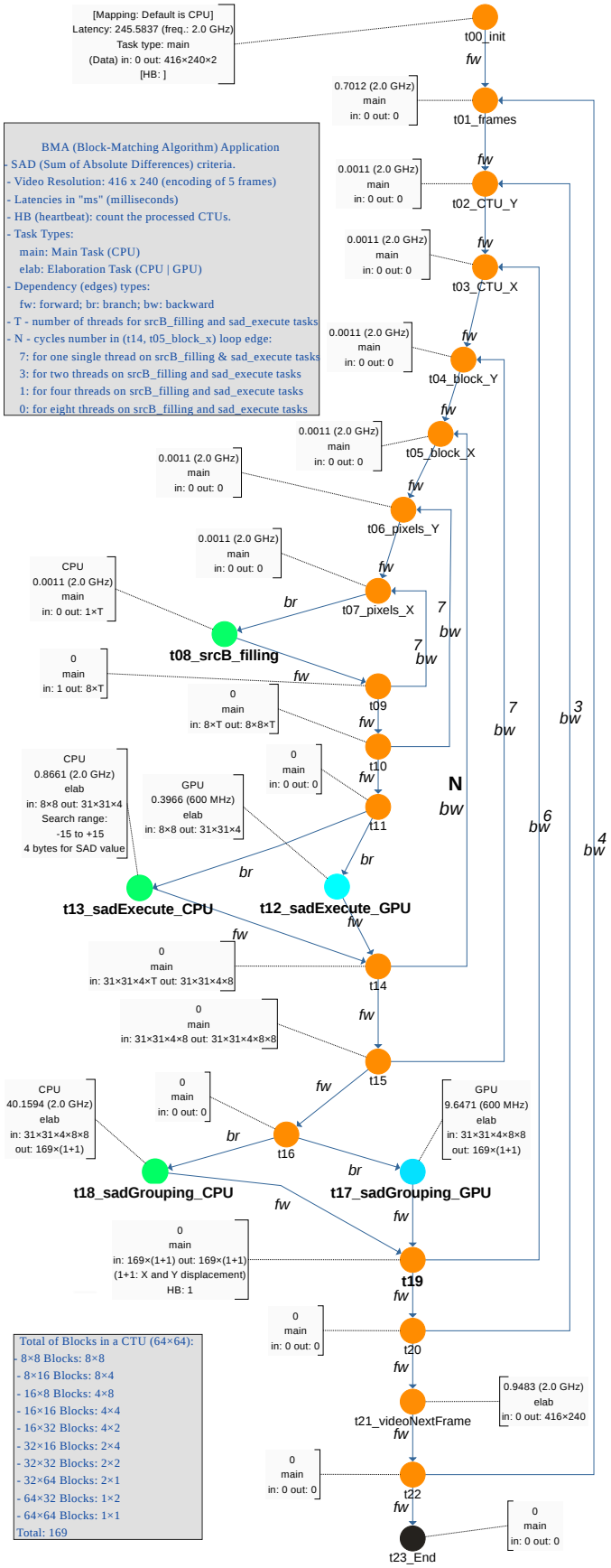


Figure 32 – BMA Application Model used as Case Study of SAVE-htlp (BETEMPS et al., 2018)

Table 3 – SAVE-htlp Experiment Configurations

#	ET1	threads	ET2	threads	ET3	threads	A15, A7, Mali
1	CPU	1	CPU	1	CPU	1	4, 4, 1
2	CPU	1	GPU	1	GPU	1	4, 4, 1
3	CPU	1	CPU|GPU	1	CPU|GPU	1	4, 4, 1
4	CPU	1	CPU|GPU	1	GPU	1	4, 4, 1
5	CPU	2	CPU+GPU	2	CPU	1	4, 4, 1
6	CPU	4	CPU+GPU	4	CPU	1	4, 4, 1
7	CPU	8	CPU+GPU	8	CPU	1	4, 4, 1
8	CPU	2	CPU+GPU	2	GPU	1	4, 4, 1
9	CPU	4	CPU+GPU	4	GPU	1	4, 4, 1
10	CPU	8	CPU+GPU	8	GPU	1	4, 4, 1
11	CPU	2	CPU	2	CPU	1	4, 4, 1
12	CPU	4	CPU	4	CPU	1	4, 4, 1
13	CPU	8	CPU	8	CPU	1	4, 4, 1
14	CPU	2	GPU	2	GPU	1	4, 4, 1
15	CPU	4	GPU	4	GPU	1	4, 4, 1
16	CPU	8	GPU	8	GPU	1	4, 4, 1
17	CPU	2	GPU	2	GPU	1	4, 4, 2
18	CPU	4	GPU	4	GPU	1	4, 4, 2
19	CPU	8	GPU	8	GPU	1	4, 4, 2
20	CPU	8	CPU+GPU	8	GPU	1	0, 8, 1
21	CPU	8	CPU+GPU	8	GPU	1	8, 0, 1
22	CPU	4	GPU	4	GPU	1	4, 4, 4

ET1: srcB_filling task; ET2: sad_execute task; ET3: sad_grouping task.
CPU|GPU: alternated CPU/GPU mapping; CPU+GPU: heterogeneous CPU/GPU mapping.

use the HTLP. According to the number of threads and PEs, threads execute in both PE types (GPU and CPU). In configurations #17 to #19, the *sad_execute* is performed considering one additional GPU to deal with the *sad_execute* and *sad_grouping* kernels. The settings #1 to #19 were simulated with the three previously cited SAVE-htlp resource mapping policies (`Base`, `A7`, and `A15`). Settings #20 to #22 were executed only with the `Base` policy. In #20 and #21, we use only A7 and A15 CPUs, respectively. In #22, we extrapolate the number of GPUs to four in the kernels' execution – four threads in CPUs and GPUs for the *srcB_filling* and *sad_execute* tasks, respectively.

Using the configuration #1 in `Base` policy as the baseline, Fig. 33 (A), (B), (C), and (D) presents an evaluation (as percentual gains) of the settings in the resource mapping policies (`Base`, `A7`, and `A15`) using the metrics Time (seconds), Performance (Frames per second), Energy (Joules), and Power (Watts) – extracted from SAVE-htlp simulations. The percentage gains represent an increase in the Performance metric meanwhile a decrease in the other metrics.

As we present in (BETEMPS et al., 2018), settings with HTLP shown better results due to the fitter matching between the tasks, its number of threads, and used PE types. Moreover, knowledge of the application characteristics allows obtaining benefits by employing different PEs for each TLP portion. It also indicates advantages in the use of heterogeneous architectures. Furthermore, executable high-level models permit, even in the early phases of the initiatives and through (rapid) system-level simulation, to evaluate candidate solutions to direct design decisions.

Moreover, we stress the importance of knowing the application characteristics to obtain benefit from each TLP portion. The BMA has three such parts: (*i*) *srcB_filling*

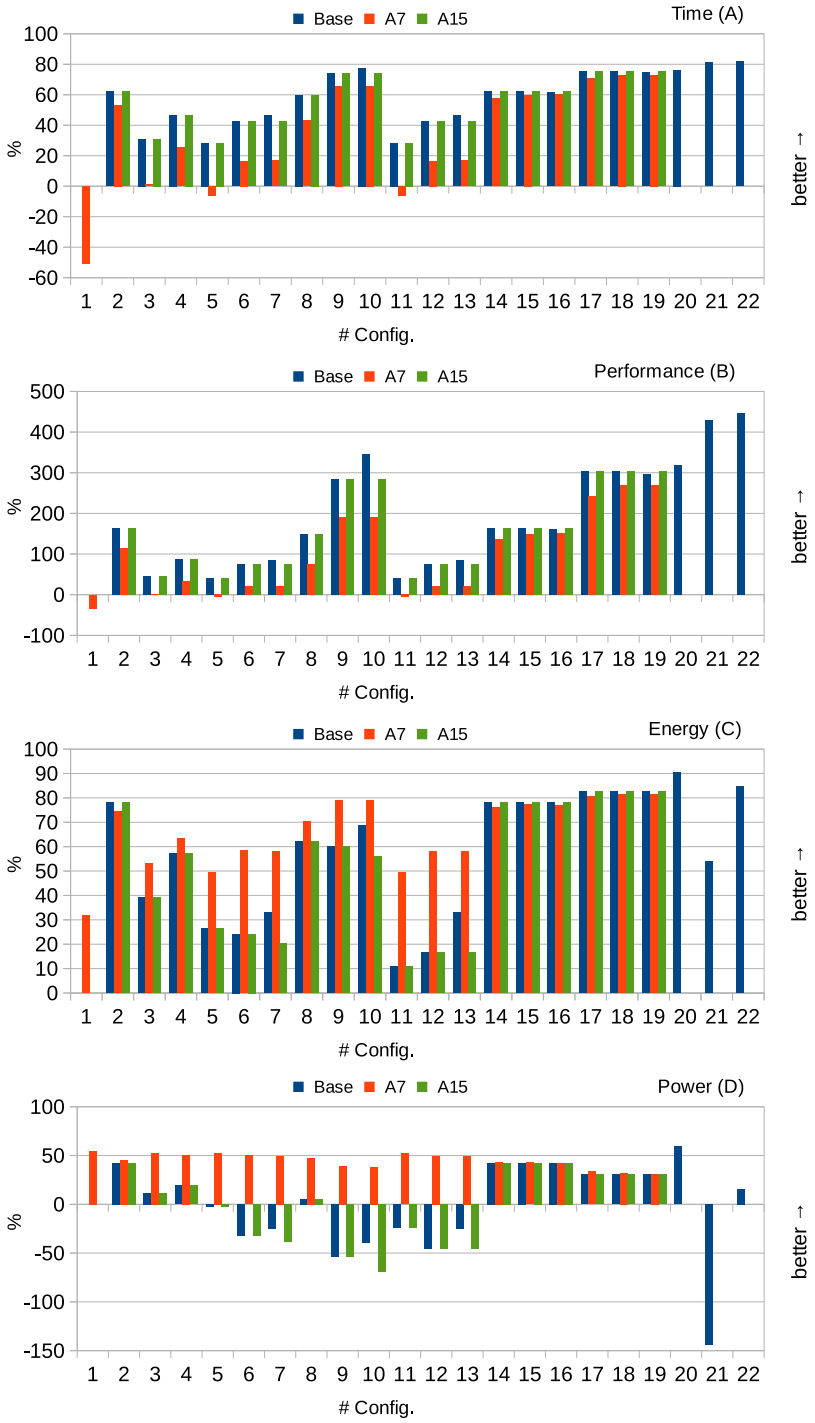


Figure 33 – Percentage gains for time (A), performance (B), energy (C), and power (D) for each configuration in the three resource policies (`Base`, `A7`, and `A15`) with #1 in the `Base` policy as baseline (BETEMPS et al., 2018).

task prepares data to be processed and can be parallelized only in CPU threads, (*ii*) *sad_execute* threads are numerous and are worth create them on GPUs and CPUs in a parallel way (HTLP), and (*iii*) *sad_grouping* have abundant data parallelism processed by the GPU sequentially.

SAVE-htlp represents the basis for the simulator implemented and used in this work – FEHetSS (Sec. 4.4). FEHetSS extends SAVE-htlp providing the capability to include FPGA devices as HW accelerators in the architecture. It also allows defining many PRRs in the device and exploring the DPR feature during the simulations. For the suitable employment of FEHetSS, the following sections present the modeling steps related to annotations in the VP models regarding the system's architecture and workload.

4.3 Modeling FPGA-enabled Virtual Platforms

Employing *VirtualHWAccel* (as an FPGA device) in System-level Simulations requires modeling several concepts of the system to include in the VP descriptions. Such modeling pass through aspects of the architecture and the workload. This section describes the modeling steps required for the simulation of FPGA-enabled Virtual Platforms. Also describing the modeling aspects of other types of PE (CPU and GPU) and the system's workload.

Fig. 34 shows the flow and tools used to estimate the annotation data required to model VPs. A VP represents the input of the FEHetSS simulator (described in Sec. 4.4). Regarding main/elaboration tasks for both CPU and GPU modeling, we use execution measurements from a real platform. Alternatively, we can generate data from cycle-accurate simulators, use information from component manufacturer datasheets, or even employ data from published research works. Based on the OpenCL programming model, we identify the application's tasks in the OpenCL source code. We insert statements to logging the tasks' latencies. Using average latency measures, we define performance models for each application's task.

Some platforms provide energy sensors (HARDKERNEL, 2020). Hence, we use a real platform to make measurements. We extract total execution time and consumed energy. Thus, we use mean values to calculate the power consumption during the execution of each application. Regarding the CPU and GPU, we create performance and power models feeding a Virtual Platforms repository.

In turn, using an FPGA as a *VirtualHWAccel* in a VP requires estimations from tools that are part of HW design flows, e.g., Quartus (INTEL, 2020b) and Intel FPGA SDK for OpenCL Software (INTEL, 2020a). FPGA-based OpenCL platforms such as Intel FPGA SDK (INTEL, 2020a) use High-level Synthesis (HLS) to produce design files and reports from which we can make estimations of power, latency, and HW resources. The

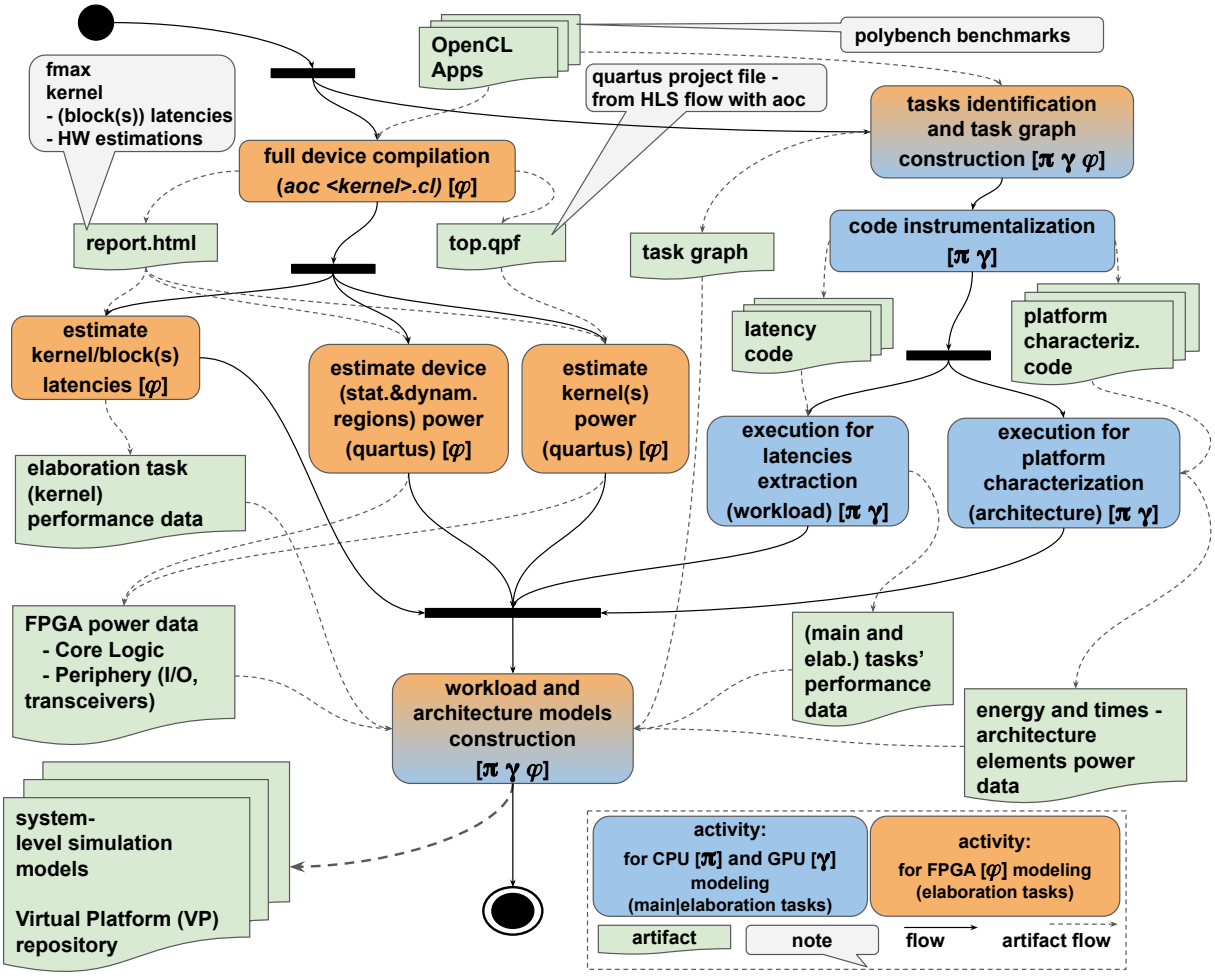


Figure 34 – Methodology for VP Modeling.

following sections present the FPGA modeling aspects regarding the architecture and the workload.

4.3.1 Architecture Modeling

Fig. 34 shows the activities related to FPGA modeling aspects. Employing OpenCL kernel codes, we use the `aoc` compiler (INTEL, 2020a) in a full device compilation. An `aoc` compilation generates several HLS design files and reports. From the reports, we can extract the maximum frequency (*fmax*) for the kernel, its required HW resources, and the latencies of each of their blocks. The synthesized files from HLS flow allow using tools obtaining power estimations – e.g., Intel Quartus (INTEL, 2020b).

Regarding the HW resources, we can use the estimated values for the applications' kernel to annotate the simulator concerning the HW utilization. Abstracting the HW resources as the number of ALUTs, we obtain the HW percentage in use by multiple HW tasks in the following way: (*i*) Obtain the number of ALUTs in the board interface ($HwBI$); (*ii*) Sum the number of ALUTs in use within each PRR P executing a HW task ($HwPRRs = \sum_{P=1}^{n} Hw\_HwTsk_P$); and (*iii*) Dividing the sum of $HwPRRs$ with $HwBI$

by the number of ALUTs available in the device ($HwDevice$) – Eq. 1.

$$HwUtilization = (HwPRRs + HwBI) \div HwDevice \quad (1)$$

4.3.1.1 Power Modeling

Since `aoc` generates HW design files from a kernel file (*.cl*), we use them to estimate power. With the Quartus tool (INTEL, 2020b), we modify the design settings of the file `top.qpf` (*quartus project file*) to make power estimations. We set the kernel frequency (in the files `top.sdc` and `base.sdc`) using the value of *fmax* (from `aoc` report). After, using the PowerPlay Power Analyzer (embedded in the Quartus tool), we estimate the power for the device using the default toggle rate (12.5%). PowerPlay reports the following estimations:

1. Transceiver Standby Thermal Power Dissipation
2. Transceiver Dynamic Thermal Power Dissipation
3. I/O Standby Thermal Power Dissipation
4. I/O Dynamic Thermal Power Dissipation
5. Core Dynamic Thermal Power Dissipation
6. Device Static Thermal Power Dissipation

The *standby power for the periphery hardware* (`SPPH`) is the sum of items 1 and 3 – Fig. 9 identifies the periphery HW of a FPGA. Summing the values of items 2 and 4 we obtain the value for *dynamic power for the periphery hardware* (`DPPH`). The estimations for the first four items are nearly constant (only item 4 suffers little variations), even in the case of several kernel being synthesized. `SPPH` and `DPPH` are device-level estimations and they fit to the device model in the simulation. Item 6 (*Device Static Thermal Power Dissipation*) express the *standby power for the device* (`SPD`). Regarding the item 5, it is the *dynamic power for the core* (`DPC`), i.e., the dynamic power during PRM's execution when it is configured in a PRR.

As for the cited estimates, Alg. 2 shows the steps to assess the power consumption of a PRR partitioned FPGA. A device can stand in four states: (i) $PowerUp\_Mode$, (ii) $Configuration\_Mode$, (iii) $User\_Mode$, and $Wait$ (Fig. 39-A). When in $User\_Mode$, any given PRR p can be in three states: (i) $PRR\_Executing$, (ii) $PRR\_Configuring$, and (iii) PRR_Idle (Fig. 39-C).

In $PowerUp\_Mode$, we model the device consuming only the standby (static) power for periphery and fabric HW resources. During a full configuration, we consider a blank bitstream configuring the entire device and its PRRs. For the blank bitstream, we

Algorithm 2 Computing Power Consumption

Require: f: FPGA object representing the device and its features
Require: t: TASK object representing a **HW task** possibly executing on a PRR
if $(sc\_time\_stamp() == SC\_ZERO\_TIME)$ **then**
  return 0.0
else if $(f.State == PowerUp\_Mode)$ **then**
  return $f.SPD + f.SPPH$
else if $(f.State == Configuration\_Mode)$ **then**
  return $f.SPPH + f.DPPH + f.configPower()$
    ▷ configPower() - set and return $f.SPD$ and $f.DPC$ during a configuration
else if $(f.State == User\_Mode)$ **then**
  $pwr = 0.0$
  pwr += $f.SPPH + f.BIHW \div f.HW \times f.SPD$
    ▷ static power for periphery resources (PH) and board interface (BI) HW
  for each PRR p of f **do** ▷ evaluates each device's PRR
    if $(p.State == PRR\_Executing)$ **then**
      pwr += $t.HW \div (t.HW + f.BIHW) \times t.Power$
        ▷ Dynamic Power for a PRR, without the BI Dynamic Power
      pwr += $(p.HW - t.HW) \div f.HW \times f.SPD$
        ▷ static power for the unused HW in the PRR
      pwr += $(t.HW) \div f.HW \times t.StaticPower$
        ▷ static power for the used HW in the PRR
    else if $(p.State == PRR\_Configuring)$ **then**
      return $p.configPower()$
    else if $(p.State == PRR\_Idle)$ **then**
      if (p is configured for some task t) **then**
        pwr += $(p.HW - t.HW) \div f.HW \times f.SPD$
          ▷ static power for the HW in the PRR not used in the core logic (PRM)
        pwr += $t.HW \div f.HW \times t.StaticPower$
          ▷ static power for the HW in the PRR used in the core logic.
      else ▷ p configured according the initial full (blank) bitstream
        pwr += $p.HW \div f.HW \times f.SPD$
          ▷ static power (of the blank bitstream) for the unused HW in the PRR
      end if
    end if
  end for
  if ((some PRR p is Executing) OR (some PRR p is (re)Configuring)) **then**
    pwr += $f.DPPH$
      ▷ Dynamic Power of the periphery HW (PH) resources
    if (some PRR p is Executing) **then**
      Let hp be the highest dynamic power of the running HW tasks
      Let hpP be the PRR configured with this highest dynamic power task
      pwr += $f.BIHW \div (f.BIHW + hpP.HW) \times hp$
        ▷ Dynamic Power of the Board Interface (BI) resources
    end if
  end if
  return pwr
end if

use an "empty" kernel (only its prototype – kernel's name and its arguments) to generate the required annotations (power estimations). During a $(re)configuration$, the periphery HW maintains static and dynamic power consumption. Regarding the configuration of a PRR or even the full device, the fabric core follows the Medium Grained Model presented by Bonamy et al. (BONAMY et al., 2014) – abstracted as the function $configPower()$ (Alg. 2). During a configuration process, this model stands that the power consumption of the region is related to the static power values of before ($P_{previous}$) and after (P_{next}) the configuration, beyond the dynamic power of the configuration controller ($P_{controller}$). Moreover, the model uses the bitstream size – that configures the region – and the time to transfer a 32-bit word – as a transferring time unit. With the bitstream size (in bytes) and the bandwidth of the configuration controller, we can calculate the time for the configuration of a 32-bit word and the whole bitstream. Based on

(BONAMY et al., 2014), Eq. 2 shows the calculation of power during the configuration of the W_{th} word (regarding the total configuration time $T_{(Re)Config}$):

$$P(W_{th}) = P_{fpga} + P_{previous} + P_{controller} + \\ (P_{next} - P_{previous}) \times W_{th} \div T_{(Re)Config} \quad (2)$$

Equation 2 produces an interpolation between the static power in points immediately before and after the configuration. P_{fpga} represents the device's remaining power consumption, e.g., the periphery HW in a full device configuration or even the consuming power in other PRRs that are executing an application kernel (or are in idle) while a specific PRR is in a configuring state ($PRR\_Configuring$).

Alg. 2 uses abstractions (like objects in object orientation) to represent the FPGA (f), the PRRs on it (p), and tasks in execution on a given PRR (t). The FPGA abstraction (f) has attributes about its operation $State$, power consumption ($SPPH$, SPD, $DPPH$, and DPC), and HW resources – for the device as a whole (HW) and the board interface ($BIHW$). In this case, the number of ALUTs denotes the HW resources. Similarly, the device's PRR (p) provides its features such as operation $State$, HW resources (number of ALUTs in a PRR – HW), and the function to estimate the power consumption during a (re)configuration ($configPower()$). Regarding the HW tasks executing in a PRR, the object t maintains information about the number of ALUTs (HW) required by the PRM in the execution of t, the value for dynamic ($Power$) and static ($StaticPower$) power considering only the PRM.

4.3.2 Workload Modeling

We use OpenCL applications to define workload scenarios. The *tasks identification and task graph construction* activity (Fig. 34) demand the analysis of the codes and identification of relevant steps in the OpenCL programming model. Due to OpenCL programming model, there are some similarities across OpenCL applications. We analyze the code structure and model it using a task graph. Fig. 35 shows a model of OpenCL applications, highlighting the identified tasks and their types (*Main* and *Elaboration*), beyond the edges and its variety (*forward*, *backward*, and *branch*). In general, the model depicts the structure of OpenCL applications. Usually, there are *backward* edges in application models where attribute `N` represents the number of additional cycles (defining a loop).

Based on the application's structure, we modify its code aiming to log the tasks' latencies (*code instrumentalization* in Fig. 34) and execute the applications (*execution for latencies extraction - workload*) in a real platform. We use average values of ten executions to characterize the task latencies (at a range of frequencies). In this manner, we define performance models for the application tasks that execute in CPUs or GPUs.

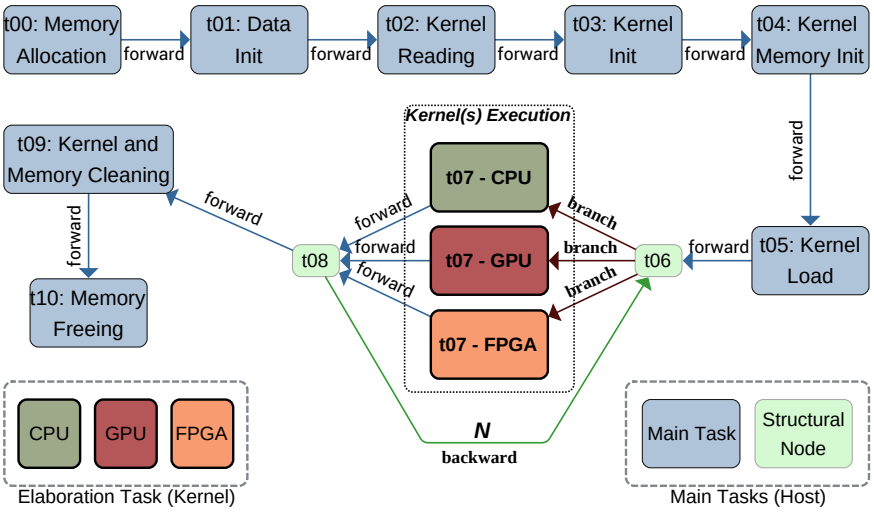


Figure 35 – Model for OpenCL Applications considering CPU, GPU, and FPGA as Processing Elements.

4.3.2.1 FPGA Kernel Latency

Using the `aoc` compiler to generate simulation files, we can employ ModelSim (MODELSIM, 2020) to simulate the kernel execution. This simulation produces waveform log files (*.wlf*) suitable to evaluate the kernel execution behavior and its latency. The analysis of simulation logs (signals in the *.wlf* file) allows estimating kernels' latency precisely. We can use specific generated signals like *start_kernel* and *completed_items* to determine the required number of cycles per item during a kernel execution. However, this strategy demands a long time involving compiling for simulation and simulating the kernels itself, being time feasible only with small size inputs. Thus, aiming to use a faster estimation procedure, we adopt another strategy, as follows.

Regarding HW tasks that execute in FPGA devices, we employ estimations based on reports provided by `aoc` compiler (Fig. 34). Fig. 36 shows the *System Viewer* portion of an kernel report. A kernel source code (*.cl*) contains a set of kernels (*K*). A set of blocks (*B*) composes each kernel *k*. Each block *b* in a Kernel *k* has its latency (L_b). A block *b* may contain a loop with I_b iterations. Also, a block has its own thread capacity TC_b . A kernel *k* may have WI_k work-items to execute. The kernel latency L_k of a kernel *k* is given by Eq. 3 using the number of cycles (C_k) executed by a kernel *k* and its operation frequency (F_k). The number of cycles executed by a kernel *k* (C_k) is given by Eq. 4, regarding the number of work-items (WI_k) that each kernel *k* must execute and the number of cycles per item consumed by kernel *k* (CI_k). Eq. 5 calculates the number of cycles per work-item (CI_k) of a kernel *k*. To calculate CI_k, we need the number of consumed cycles per each block *b* (CB_b in Eq. 6) of a kernel *k* and the thread capacity of block *b* (TC_b). The consumed cycles per block *b* (CB_b in Eq. 6) considers if the block has only one iteration or a loop. It is worth mentioning that in the case of using multiple compute units in a kernel's implementation, the number of work-

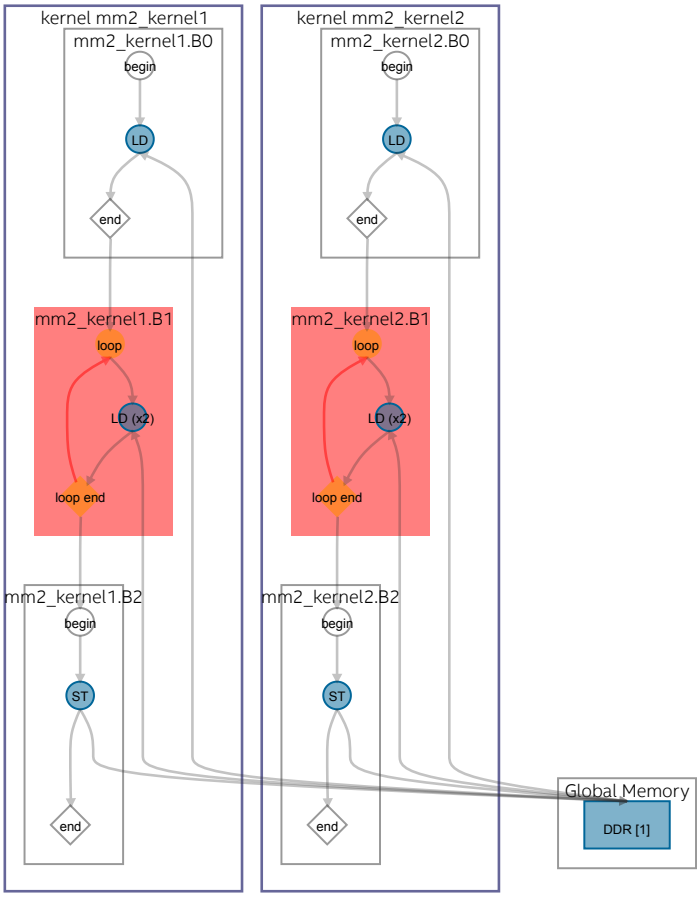


Figure 36 – System Viewer portion of an kernel Report.

items to be processed by each compute unit will be decreased by a ratio proportional to their number. Moreover, when employing loop unrolling, the number of iterations of a kernel block containing unrolled loop will diminish according to its unrolling factor.

$$L_k = C_k \div F_k \tag{3}$$

$$C_k = \lceil \sum_{k=1}^{|K|} WI_k \times CI_k \rceil \tag{4}$$

$$CI_k = \sum_{b=1}^{|B|} CB_b \div TC_b \tag{5}$$

$$CB_b = \begin{cases} (L_b + WI_k) \div WI_k & I_b = 1 \\ I_b \times L_b & \text{otherwise} \end{cases} \tag{6}$$

4.4 FEHetSS – FPGA-Enabled Heterogeneous System Simulator

This section describes the FPGA-Enabled Heterogeneous System Simulator (FEHetSS). FEHetSS was built on top of SAVE-htlp Simulator (BETEMPS et al., 2018), which is an extension of the SAVE simulator (MIELE et al., 2015).

FEHetSS is written in SystemC using TLM (Transaction Level Modeling) as abstraction level (CAI; GAJSKI, 2003). It uses task graphs to model the workload (applications) – including annotations of performance model (e.g., frequency and latency) – and a set of generic (HW) resources to set up the architecture – describing the PE type and its power model (frequency, power, and idle power). In this work, we include a *VirtualHWAccel* – FPGA unit – that simulates the execution of an kernel as performed by a HW accelerator. In FEHetSS, a *VirtualHWAccel* also features the capacity to model PRR portions and considers DPR during the simulations.

FEHetSS receives an XML file containing a *Virtual Platform* which describes the workload (applications as task graphs) and the architecture (architectural blocks) models. Fig. 37 presents an UML class diagram (GOMAA, 2011) that conceptually depicts the elements of the FEHetSS input. Primarily, a *Virtual Platform* consists of an *Architecture*, a *Workload*, and an *Orchestrator*. *Processor*s compose the *Architecture*, being featured by *DVFS* (Dynamic Voltage and Frequency Scaling) elements (*Power*, *Idle Power*, and *Frequency*). When the *Processor* element is a FPGA, it contains some extra annotations. First, the *Hardware Resources* available in the FPGA device in terms of ALUTs (Adaptive Look-Up Table), FFs (flip-flops), RAMs (memory blocks), and DSPs (Digital Signal Processing blocks). Secondly, the same types of HW resources set the device's board interface, i.e., the number of components in the board interface. Lastly, some *Operational Parameters* defines the FPGA operation, as follows: the latency for the device power-up, the size of the full bitstream file, the bandwidth for the (re)configuration process, the dynamic power for PR controller, and its base operating frequency.

A *Workload* includes all *Application*s. A task graph describes an *Application* containing *Node*s and *Dependencies*. A *Node* represents tasks of two types: (*i*) *Main* tasks, which are application portions always executed sequentially by the CPU (host) and representing tasks that coordinate the execution; (*ii*) *Elaboration* tasks (possibly paralleled) that represent *kernels* executed by CPU, GPU, or FPGA. *Node*s contain a *Performance Model* consisting of latency and frequency values. *Dependency* represents edges of three types: (*i*) *forward* – designates the normal application's advance, (*ii*) *backward* – represents a loop edge, and (*iii*) *branch* – indicates the advance to an elaboration task (a kernel) that can be possibly executed in distinct PEs. When a *Node* represents an Elaboration Task and its *mapping* is FPGA, a *Core Logic* element describes information about the PRM operating status in the device: power, static power,

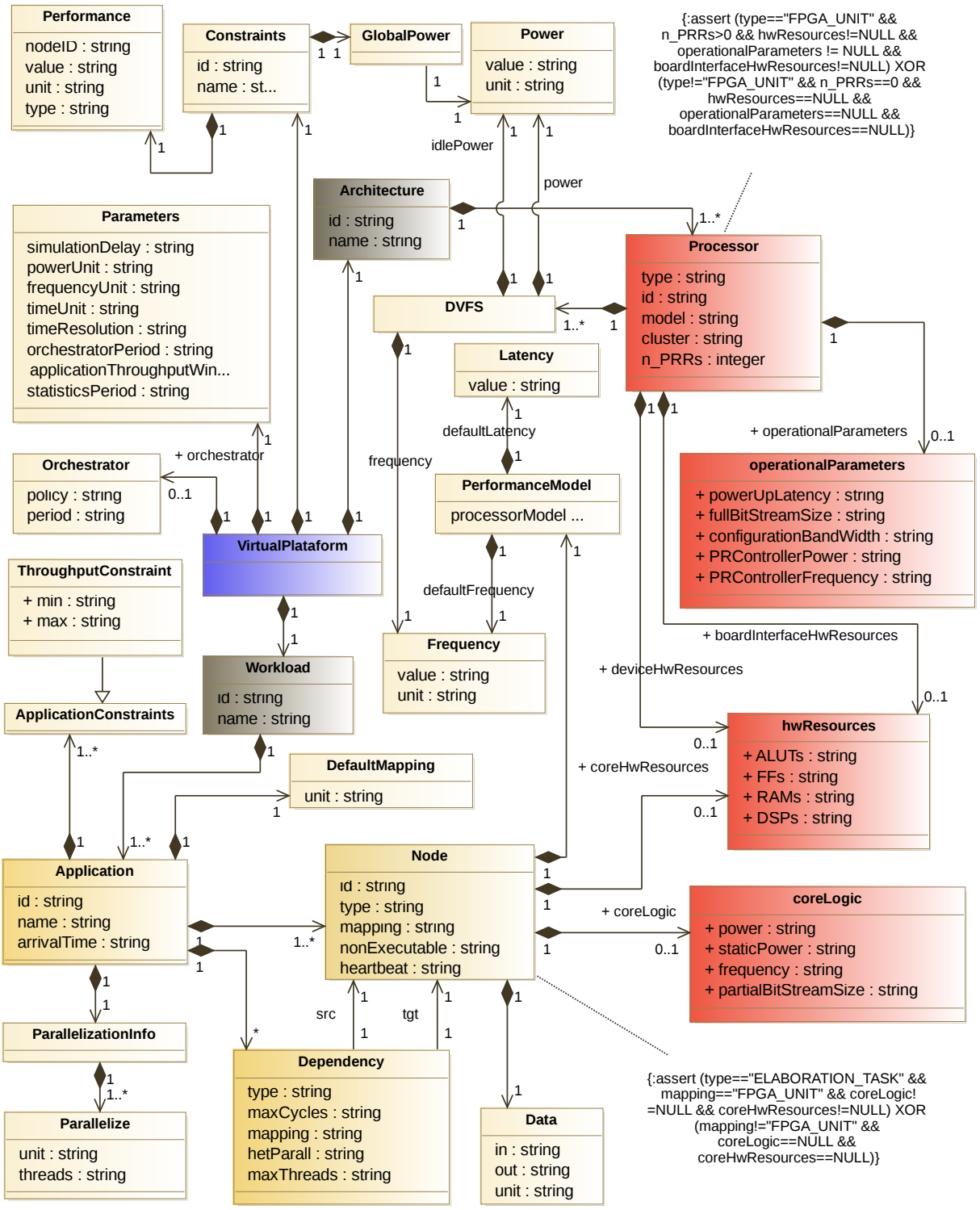


Figure 37 – FEHetSS's VP Input Elements and its Relationships – Elements of a XML File describing a VP (Architecture and Workload).

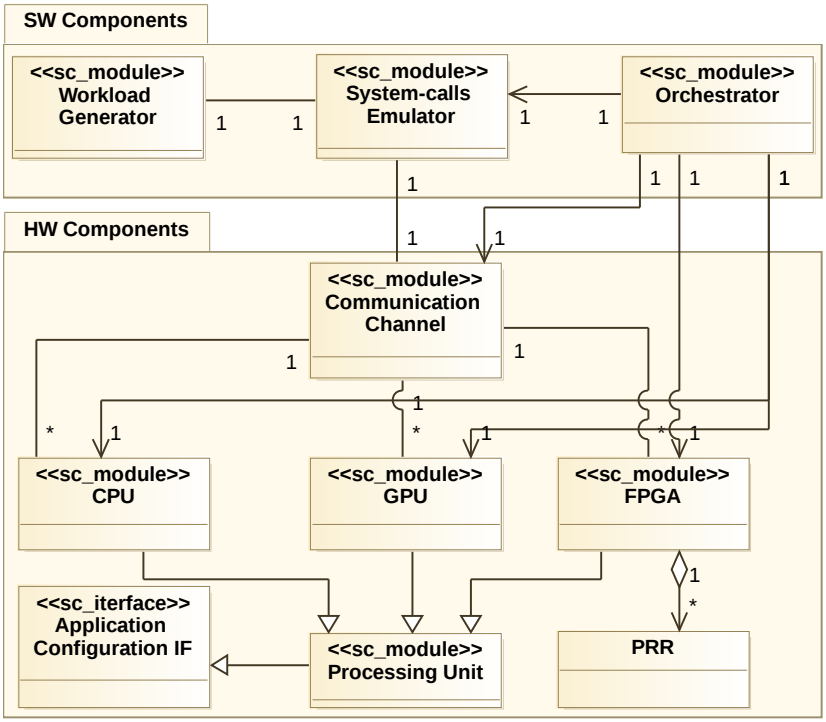


Figure 38 – Main Components of FEHetSS System-level Simulator.

frequency, and partial bitstream file's size. Further, a *Hardware Resource* element describes the number of elements in the core logic, i.e., an HW task – a kernel executed as HW accelerator in FPGAs.

Fig. 38 describes the FEHetSS main components. The XML input file sets up the components' properties. The FPGA implementation employs a SystemC module (*sc_module*), like the other PE types. Also, a *PRR* class allows representing the concept of PRRs inside a device. The *Communication Channel*, as an *sc_module*, provides a communication component representing *buses* and *memory*. Each PE has a FIFO (first in, first out) scheduler to manage the execution of the tasks' threads. In a CPU, those threads can be interrupted. The described elements represent HW components. Furthermore, FEHetSS has some SW components. *Workload Generator* uses the descriptions of the applications in the input file and acts like a system-user triggering its beginning. The component *System-calls Emulator* manages the applications executions, sending threads of the tasks to execute on the available PEs, according to the mapping inside the applications. *Orchestrator* component produces logs of the simulation including data about power, utilization, scheduling, and the simulation itself.

4.4.1 FPGA Processing Element Simulation Flow

The FPGA simulation follows the code of a module executed as a SC_THREAD, according to the state machine described in Fig. 39 - A. Initially, the device pass through a *PowerUp_Mode*, following to a full device configuration (*Configuration_Mode*) con-

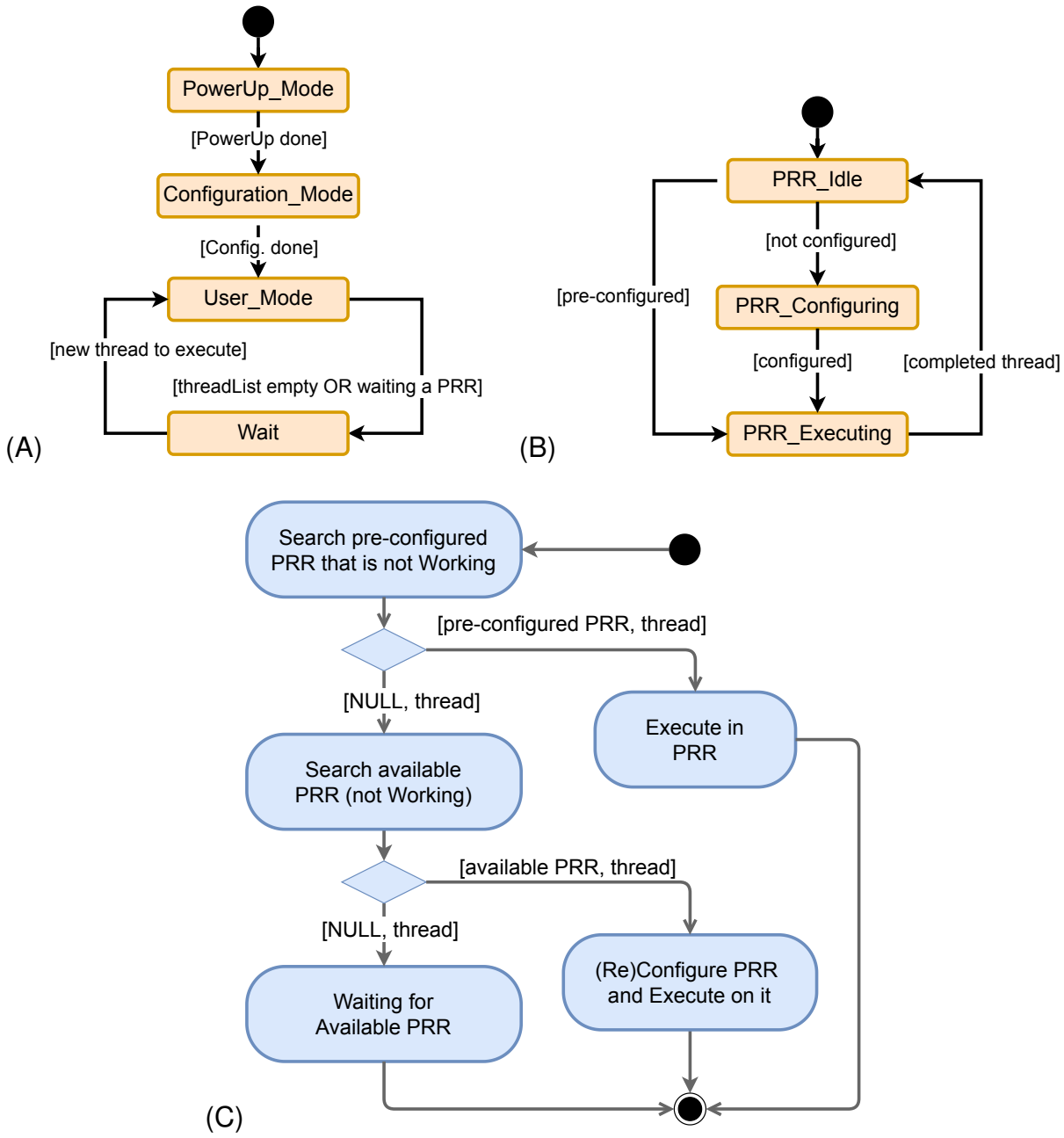


Figure 39 – (A) FPGA States. (B) PRR States. (C) User_Mode Procedure.

sidering a blank bitstream. After, there are two possible states: *User_Mode* and *Wait*. When no HW task thread is ready to execute, the device enters a *Wait* state. Existing a thread ready to execute, FPGA pass to *User_Mode*, executing its procedure (Fig. 39-C). *User_Mode* procedure executes when a new HW task thread is ready to start. First, it searches for a previously configured PRRs that is idle – if any available, it executes the thread using that PRR. Otherwise, if no (pre)configured PRR is available, it searches for an idle PRR, reconfigures it, and executes the thread. In the case of no PRR available, the thread waits. All device's PRRs, during system operation, follow the state machine in Fig. 39-B. Having a previously configured HW task, a PRR can pass to a *PRR_Executing* state. In turn, a *PRR_Configuring* state is necessary before the execution. The FEHetSS simulation is based on a VP model, as presented in Sec. 4.3.

Considering the modeling aspects described in this chapter, we can proceed (Chap. 5) describing some experiments with the FEHetSS simulator, aiming to evaluate its capabilities and the meeting of the thesis research challenges.

4.5 Chapter Summary

In this chapter, we have presented the modeling&simulation aspects of the thesis. First, we describe the SAVE-htlp simulator – presented in (BETEMPS et al., 2018). SAVE-htlp allows to model threads with the Heterogeneous Task-level Parallelism (HTLP). In HTLP, different implementations of a task (e.g., aiming CPU, GPU, and FPGA) execute in parallel, making the called Heterogeneous Task-level Parallelism.

After, we describe the modeling steps involving FPGA-Enabled architectures. It includes the modeling of the Architecture. About CPU and GPU, we use profiled data from a platform (featuring power sensors) execution using OpenCL instrumented code. Regarding FPGA modeling, we apply HW design tools providing estimations for latency and power. The HLS tool Intel FPGA SDK for OpenCL Software (INTEL, 2020a) provides reports and produces HW design files. From these reports, we collect estimations related to maximal frequency, HW resources, and block latencies. With the HW design files, we use the PowerPlay tool, integrated into Quartus (INTEL, 2020b) tool, to make power estimates. Regarding Workload modeling, we use the reports from `aoc` compiler (a tool from (INTEL, 2020a)) to extract the kernel's block latencies. Thus, with the power and latency data, we employ a power model to describe the FPGA devices and a latency estimator to define an application's HW task description capable of execution in a *VirtualHWAccel*.

Last, we present the main SystemC modules of the FEHetSS simulator. We also describe its input file's structure and the FPGA's simulation flow. FPGA simulation allows the device to act as a PE, or even several PEs if applying DPR features and multiple

PRRs. FEHetSS receives a VP as input containing models of the Architecture and the Workload. The model generation follows the methodological steps also presented in this chapter. Architecture and Workload models are available in a VP Repository. This repository makes its VP models available through a series of automation scripts.

In a large view, the VP modeling methodology and the FEHetSS simulator make up an HW&SW system design framework for the evaluation of candidate solutions in an Early DSE context. As researchers, during the development of the approach, we actuate in many branches, as follows:

- the choice of OpenCL programming model (Host and Device code) as a basis to model applications, employing the Polybench benchmark application's kernel as examples;
- the use of an HLS tool (`aoc` compiler) to provide kernel and device estimations;
- the utilization of HW design tool (quartus/PowerPlay) aiming to estimate power to the HW tasks core logic and the device itself;
- the performing of platform executions with OpenCL code to extract data about CPU and GPU power and latency;
- the preparation of the VP Repository and its VP creation scripts;
- the use of the Architecture and Workload Modeling methodology creating VP models, including the Power and Kernel/Block(s) Latency models for FPGA device; and
- the implementation of *VirtualHWAccel* making up FEHetSS simulator, including features of DPR and PRR modeling&simulation.

We can also highlight the main developed thesis' elements, which are:

- The Architecture and Workload Modeling Methodology considering different PEs, especially the *VirtualHWAccel* modeled as an FPGA featuring DPR and PRR;
- The FEHetSS simulator implementing our conceptions regarding reconfigurable PRR-partitioned FPGA devices including the account of reconfiguration time and power during DPR; and
- The VP repository containing several annotated model portions being available to create solutions based on a VP description.

5 ASSESSING FEHETSS AS A SOLUTION EVALUATOR DURING DSE AND HW DESIGN ACTIVITIES

To demonstrate the capabilities of FEHetSS, we plan four Case Studies employing the artifacts produced by applying the methodological steps (Sec. 4.3) and the FEHetSS simulator (Sec. 4.4). In this chapter, we present the first two Case Studies. In Sec. 5.1, we use the GQM approach (BASILI; CALDIERA; ROMBACH, 1994) presenting the Goal of each Case Study by defining the Question to answer and the Metrics to evaluate on each one. Sec. 5.1.1 presents the workload applications, followed by the detailing of the generation of the Architecture and Workload models (Sec. 5.1.2) by applying our methodology. Regarding the experiments, we first present a (manual) DSE Case Study defining some heterogeneous settings to simulate/evaluate with FEHetSS (Sec. 5.2). After, in Sec. 5.3, we employ FEHetSS to evaluate HW Design alternatives of an application's kernel.

5.1 Case Studies Goals and Preparation

Tab. 4 resumes the planned Case Studies evaluating FEHetSS and the modeling methodology. We use the GQM (Goal, Question, Metric) Approach (BASILI; CALDIERA; ROMBACH, 1994) to describe each Case Study and its scenarios by characterizing its `Goal`, what `Question` we want to answer, and which `Metric`(s) we use in each case. In the metrics column, we insert plot types aimed to show the results.

5.1.1 Experiment Workload

In the experiments, we use the Polybench benchmark (POUCHET, 2012) modeling all its applications (1 to 15). Polybench (POUCHET, 2012) is written in C/C++ language with OpenCL kernels. Tab. 5 describes some features of each application – including its *category*, the *arrays size*, the *number of kernels* (# K), and the *index space* dimensions. Fig. 35 (in Sec. 4.3.2) shows the Polybench applications structure – modeled as from OpenCL codes. Most of the applications do not exhibit loops outside the kernels (*backward* edge with `N` equal to zero). However, three applications comprise it

Table 4 – Summary of Case Studies #1 and #2

Case Study	Scenario	Goal	Question	Metric(s)	Architecture Type	Workload Appl.	Section
#1	1	Evaluate the FEHetSS's Capability during a (manual) DSE of Heterogeneous Architectures (settings), considering high-level models produced basing the OpenCL programming model.	Based on high-level models, can FEHetSS provide appropriate log metrics in heterogeneous architectures' evaluation during a (manual) DSE?	- System's Time, Energy, and Utilization, and Simulation's Time. Graphs: 3D Scatter plot and Line chart.	- Homogeneous and Asymetric (only CPU) - Heterogeneous (CPU and GPU) - Heterogeneous (CPU and GPU and FPGA) - Heterog. (CPU and FPGA) with Optimizations	All Polybench Appl. (1-15)	5.2
#2	1	Evaluate the FEHetSS's Capability while assessing HW Tasks Design Alternatives regarding high-level models describing the FPGA device and different HW implementations for an application's kernel.	Can FEHetSS be used to assess different HW task implementations based on high-level models describing the tasks themselves and the HW accelerator device (FPGA)?	- System's Time, Energy, and Utilization, and Simulation's Time. Graphs: 3D Scatter plot and Line chart.	Heterogeneus (FPGA)	3	5.3
#2	2	Evaluate the FEHetSS's Capability while assessing HW Tasks Design Alternatives regarding high-level models describing specific FPGA devices with different resources and distinct HW implementationsOverview for an application's kernel.	Can FEHetSS be used to assess different HW task implementations based on high-level models describing the tasks themselves and the HW accelerator devices (FPGA) with diversified resources?	- System's Time, Energy, and Utilization. Graphs: 3D Scatter plot with Pareto Front regarding HW Task Implementation and Specific FPGA Device.	Heterogeneus (FPGA)	3	5.3

Table 5 – Polybench/GPU (OpenCL) Applications (POUCHET, 2012)

ID	App	Description	Category	Array Size	Data-Type	# K	Index Space (x,y)	# Loops Outside Kernel(s)
1	2DCONV	2D Convolution	Convolution	4096	float	1	(32,8)	1
2	3DCONV	3D Convolution	Convolution	256	float	1	(32,8)	254
3	2MM	2 Matrix Multiplications (d=a.b; e=c.d)	Linear Algebra	2048	float	2	(32,8)	1
4	3MM	3 Matrix Multiplications (e=a.b; f=c.d; g=e.f)	Linear Algebra	512	float	3	(32,8)	1
5	ATAX	Matrix Transpose and Vector Multiplication	Linear Algebra	4096	float	2	(256,1)	1
6	BICG	BiCG Sub Kernel of BiCGStab Linear Solver	Linear Algebra	4096	float	2	(256,1)	1
7	GEMM	Matrix-multiply (C=alpha.A.B+beta.C)	Linear Algebra	512	float	1	(32,8)	1
8	GESUMMV	Scalar, Vector and Matrix Multiplication	Linear Algebra	4096	float	1	(256,1)	1
9	GRAMSCHM	Gram 4 -Schmidt decomposition	Linear Algebra	2048	double	3	(256,1)	2048
10	MVT	Matrix Vector Product and Transpose	Linear Algebra	4096	double	2	(256,1)	1
11	SYR2K	Symmetric rank-2k operations	Linear Algebra	2048	float	1	(32,8)	1
12	SYRK	Symmetric rank-k operations	Linear Algebra	1024	float	1	(32,8)	1
13	CORR	Correlation Computation	Datamining	2048	float	4	† (256,1), (32,8)	1
14	COVAR	Covariance Computation	Datamining	2048	float	3	‡ (256,1), (32,8)	1
15	FDTD-2D	2-D Finite Different Time Domain Kernel	Stencils	2048	float	3	(32,8)	500

†: (256,1) for kernel 1, 2 and 4. (32,8) for kernel 3. ‡: (256,1) for kernel 1 and 3. (32,8) for kernel 2.

with the respective value for `N`: (2 - 3DCONV) 253; (9 - GRAMSCHM) 2047; and (15 - FDTD-2D) 499.

5.1.2 Generating Architecture and Workload Models

Considering the methodological steps described in Sec. 4.3 for CPU and GPU modeling, we perform the measures of latency and power in a real platform. We use the Odroid-XU-3 platform featuring the Exynos 5422 MPSoC (HARDKERNEL, 2020). This MPSoC features a big.LITTLE architecture – clusters with A15 and A7 quad-core ARM processors – as well as a MALI-T628 MP6 GPU, also providing energy sensors for A7 and A15 cores, Mali GPU, and memory. For idle power and (operation) power measurement, we proceed to alter the CPU affinity during the application's execution (command *taskset*). Thus, we use real platform executions (at least ten times per configuration) to obtain a power model for these PE types. The frequency range for the CPU is 400 to 1400 MHz for the A7 core and 400 to 2000 MHz for the A15 core. In the

Table 6 – Arria10GX Product Family, Available Resources, Powers, and Bitstream

Device Model *Arria10*	*GX 160*	*GX 220*	*GX 270*	*GX 320*	*GX 480*	*GX 570*	*GX 660*	*GX 900*	*GX 1150*
ID (VP)	9	8	7	6	5	4	3	2	1
HW Resources:									
Logic Elements (K)	160	220	270	320	480	570	660	900	1150
ALUTs	123020	160660	203240	239800	367180	434160	503360	679240	854400
FFs	246040	321320	406480	479600	734360	868320	1006720	1358480	1708800
RAMs	440	587	750	891	1431	1800	2131	2423	2713
DSPs	156	192	830	985	1368	1523	1687	1518	1518
Transceivers	12	12	24	24	36	48	48	96	96
GPIO	288	288	384	384	492	696	696	768	768
Power (PW) and Idle Power (IPW):									
Core Fabric IPW (W)	0.5399	0.7051	0.8920	1.0525	1.6116	1.9055	2.2093	2.9812	3.7500
Core Fabric PW (W)	0.3456	0.4513	0.5709	0.6736	1.0314	1.2196	1.4139	1.9080	2.4000
Periphery HW IPW (Transc. + IO) (W)	0.8328	0.8328	1.3518	1.3518	1.8904	2.5858	2.5858	4.1520	4.1520
Periphery HW PW (Transc. + IO) (W)	1.6028	1.6028	2.5965	2.5965	3.6283	4.9646	4.9646	7.9500	7.9500
Bitstream Size:									
Full Device (bytes)	24940599	32571587	41204091	48616124	74440652	88019918	102049258	137706488	173217748

case of GPU, we use only the (max) 600 MHz frequency. Figures 40 and 41 present the values, used in the VP models, for the Power (A) and Idle Power (B) of the A7 and A15 cores, respectively. Besides, Fig. 42 presents the Power and Idle Power for the MALI GPU (at 600MHz frequency).

For the FPGA device modeling, we choose a specific device family for the case studies – Arria10GX (INTEL, 2018) – using Intel FPGA SDK for OpenCL Software (INTEL, 2020a) and Quartus (INTEL, 2020b) as supporting tools to perform the power/latency estimations. According to (INTEL, 2018), the Intel Arria 10 devices are ideal for high-performance, power-sensitive, midrange applications in diverse markets. We generate the estimations using the specific device Arria10GX1150 (Arria10 - product line GX 1150). However, we generate power estimates for all the other specific devices in the Arria10GX family. The power estimates consider the values obtained for GX 1150 device and proportionally calculates the values regarding the number of each type of resource available in the respective product.

Since it is mandatory to annotate the FPGA model with the size of bitstreams, we use the size of the configuration files generated by the `aoc` compilation (*.aocx* file). It indicates the bitstream size concerning the initial full (blank) configuration regarding the Arria10GX1150 product (Tab. 6). We calculate the bitstream size for the other devices taking as a basis the latter (Arria10GX1150) and considering the number of ALUTs in each specific product, proportionally calculating the size of the bitstreams. Further, for each PRR, we use the size of the (*.aocx*) files built for a single kernel aiming simulation with ModelSim – considering that this file contains only the information of the simulated HW. Associated with the bitstreams is the PR controller. We use the expected bandwidth for the Intel PR controller (INTEL, 2020b) to meet the VP model – maximum frequency of 100MHz using 32-bit words (400 MB/s).

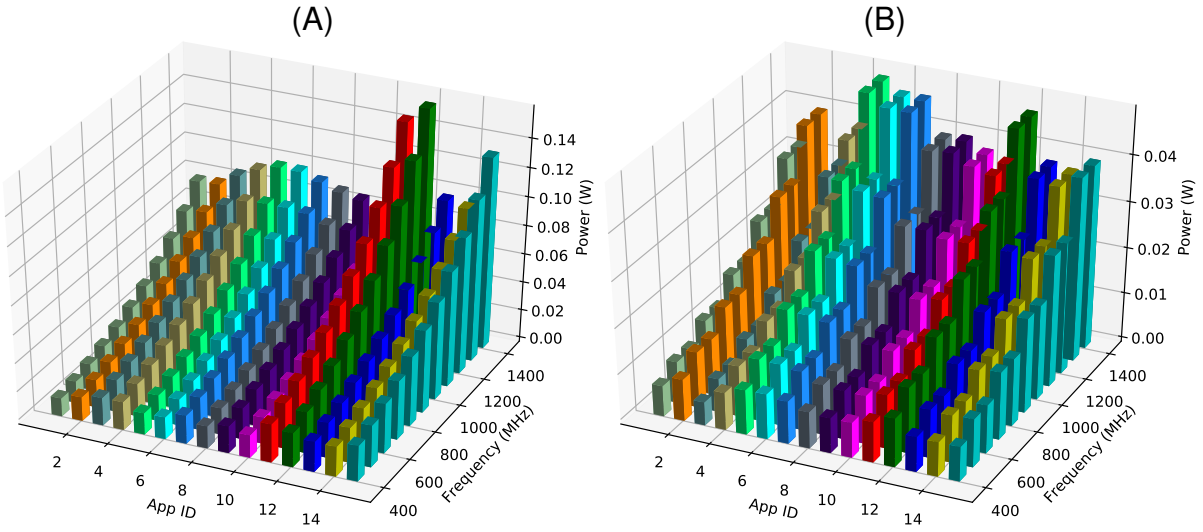


Figure 40 – A7 Core: (A) Power and (B) Idle Power (by Application and Frequency).

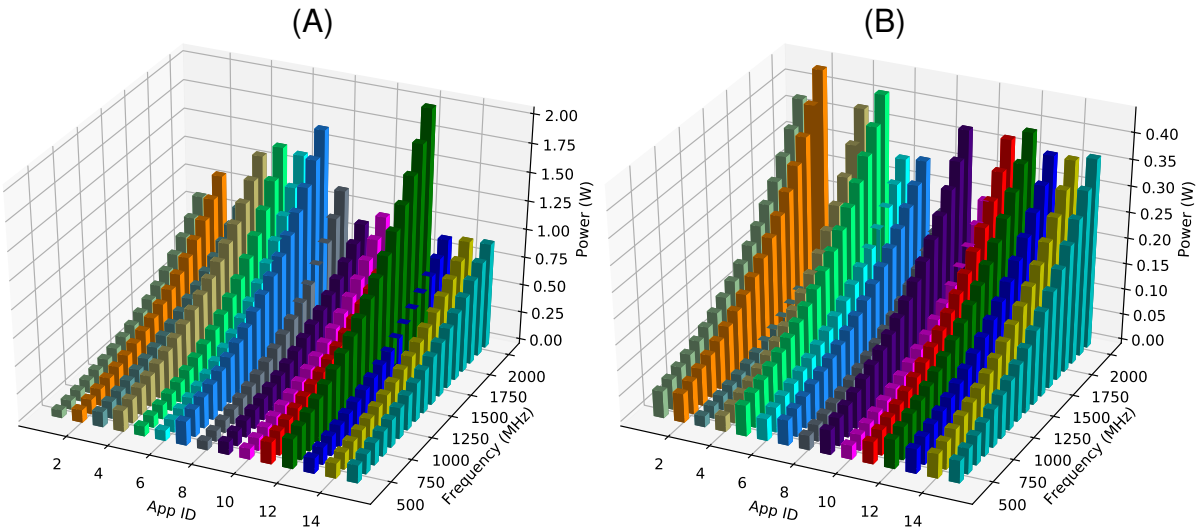


Figure 41 – A15 Core: (A) Power and (B) Idle Power (by Application and Frequency)

Tab. 6 resume the devices' features showing the product name, the available HW resources, the estimated powers for the whole device (and considering a blank bitstream) given by the supporting tools, and the bitstream sizes for each Arria10GX specific device.

Regarding the workload modeling, we use the OpenCL codes of the Polybench (POUCHET, 2012) in its original form. These implementations do not employ any optimization in the code. Using the `aoc` compiler from Intel FPGA SDK for OpenCL Software (INTEL, 2020a), we generate reports, bitstream, and other design files for each Polybench application. Following the steps described in Sec. 4.3.2.1, we calculate the kernels' latencies of each application of the Polybench. Moreover, we estimate the kernel's latencies considering some optimizations. Tab. 7 resumes the estimated latencies. For all applications, we estimate the latency and the respective powers for the core (module's implementation of a kernel), also estimating the core's HW resources

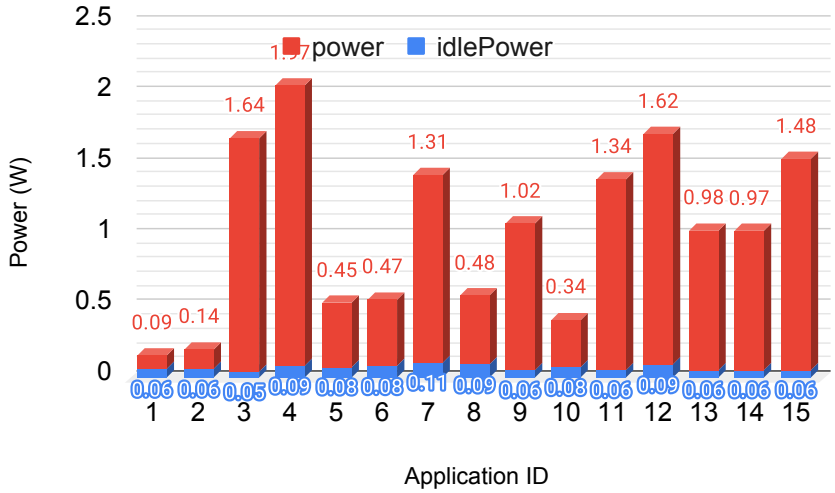


Figure 42 – MALI-T628 Power and Idle Power (by Application at 600 MHz)

Table 7 – HW Tasks' Implementations of the Workload's Applications

ID	Description	Loop Unrolling	Compute Unit	Applications
1	Original Kernel(s) Code – one Compute Unit and without Loop Unrolling.	No	1	All
2	Two (2) Loop Unrolling (LU), applied in the kernel's loop (when present) and only in the most internal in the case of nested loops.	2	1	All
3	Two (2) Compute Units (CU).	No	2	All
4	Four (4) Loop Unrolling (LU), applied in the kernel's loop (when present) and only in the most internal in the case of nested loops.	4	1	2MM(3)
5	Four (4) Compute Units (CU).	No	4	2MM(3)
6	Eight (8) Loop Unrolling (LU), applied in the kernel's loop (when present) and only in the most internal in the case of nested loops.	8	1	2MM(3)
7	Eight (8) Compute Units (CU).	No	8	2MM(3)

and the size of the partial bitstream. Besides, we consider three variants of the code: original, with (two) loop unrolling optimization, and using two computing units in the HW task implementation (according to Tab. 7). Tables 8, 9, and 10 show the features of the Partial Reconfigurable Modules associated with the mentioned code variants. Furthermore, for application 2MM (3), we estimate the same elements considering the Loop Unrolling optimization with four (4) and eight (8) as unrolling factor; and the multiple Compute Unit optimization with also the values four (4) and eight (8). Tab. 11 present these elements.

To perform the FEHetSS simulations in the simulations of Case Studies #1 and #2, we use the following computer: A notebook with four Intel Core i5-7200U CPUs @ 2.5GHz, 64 bits, 4GB memory, running Linux Ubuntu 18.04.5 LTS Operating System. Whenever possible, we use several FEHetSS threads according to the number of available processing cores.

5.2 Case Study #1 – DSE of Heterogeneous Settings

Tab. 12 presents the Architecture Scenarios used in the first Case Study – configurations 1 to 10. Regarding the workload, we use the original code of the Polybench

Table 8 – HW Tasks' Latencies, Powers, and Partial Bitstreams - Original Codes

ID	Appl.	Freq. (MHz) (fmax)	Latency (us)	Latency (cycles)	Core Dynamic Power	Core Static Power	Core ALUTs	Core FFs	Core RAMs	Core DSPs	Partial Bitstream
1	2DCONV	357.95	46870.91	16777441	4.2968	4.0533	11824	21317	83	13	10059016
2	3DCONV	287.50	229.14	65877	4.3516	4.0823	10070	36424	188	28	11652168
3	2MM	316.40	54072518.60	17108544885	8.5548	4.9577	19584	36902	256	11	12064052
4	3MM	307.29	1308736.79	402161727	6.0787	4.3923	29214	54972	374	16	13140096
5	ATAX	319.44	104542.57	33395079	4.6038	4.1124	13862	28496	196	5	11818240
6	BICG	306.81	108846.62	33395231	4.4123	4.0819	15173	32433	208	5	11822072
7	GEMM	301.13	445392.16	134120940	3.7126	3.9726	10984	21864	155	7	11141480
8	GESUMMV	328.12	50965.66	16722853	4.9995	4.1685	11110	29095	198	5	11656760
9	GRAMSCHM	301.13	41649.81	12542007	6.4463	4.4407	43561	63920	347	49	16182832
10	MVT	321.42	103952.78	33412502	5.1336	4.1934	22504	34720	198	11	12828428
11	SYR2K	290.17	29512170.46	8563546503	4.0922	4.0414	17143	31532	242	10	11271148
12	SYRK	306.25	3496088.35	1070677058	3.7732	3.9702	11030	21821	156	7	11072096
13	CORR	285.15	15048366.82	4291041798	6.8479	4.5287	33279	76139	432	30	15607692
14	COVAR	302.08	5777583.75	1745292498	5.4510	4.2415	19981	46246	255	14	13388068
15	FDTD-2D	335.22	37537.96	12583476	6.3249	4.4029	23562	49406	193	19	13496324

Table 9 – HW Tasks' Latencies, Powers, and Partial Bitstreams - Two Loop Unrolling

ID	Appl.	Freq. (MHz) (fmax)	Latency (us)	Latency (cycles)	Core Dynamic Power	Core Static Power	Core ALUTs	Core FFs	Core RAMs	Core DSPs	Partial Bitstream
1	2DCONV	357.95	46870.91	16777441	4.2968	4.0533	11824	21282	81	13.5	10059016
2	3DCONV	287.50	229.14	65877	4.3434	4.0812	10070	36357	186	28.5	11652168
3	2MM	275.57	31027527.59	8550199363	4.7363	4.1472	26258	46389	338	16.0	12531588
4	3MM	275.00	733001.32	201575362	6.1897	4.4450	39297	69672	498	24.0	13140096
5	ATAX	312.50	53400.28	16687589	4.7411	4.1349	14659	31996	210	8.0	11818240
6	BICG	329.86	50589.91	16687645	5.1744	4.1925	15958	35899	219	8.0	11822072
7	GEMM	290.63	168574.54	48991975	3.8690	3.9954	14312	26573	195	11.0	11141480
8	GESUMMV	319.44	26150.05	8353489	4.9626	4.1703	11927	30557	209	7.5	11656760
9	GRAMSCHM	278.13	22613.15	6289282	7.5715	4.7117	68075	96376	523	66.0	16182832
10	MVT	335.94	49720.75	16703063	6.3340	4.3976	32281	44432	218	20.0	12828428
11	SYR2K	300.00	14278239.90	4283471971	4.5353	4.1118	18557	29851	249	15.5	11271148
12	SYRK	307.29	1743713.55	535828643	3.9421	4.0050	11724	20996	155	9.5	11072096
13	CORR	287.50	2402291.15	690658705	7.6481	4.6896	35301	86590	477	39.0	15607692
14	COVAR	322.92	2554513.89	824895109	6.7041	4.4657	21441	54031	288	19.0	13388068
15	FDTD-2D	335.22	37537.96	12583476	6.3249	4.4029	23562	49339	191	19.0	13496324

suite, triggering all 15 applications in the arrival time 0 (zero) during the simulation. To evaluate optimization alternatives, implemented through pragmas in OpenCL code, we also simulate VPs considering two optimization strategies (settings 9 and 10 in Tab. 12). The first one aims to raise the parallelism using two computing units in the kernels' execution (*__attribute__((num_compute_units(2)))*). The other optimization focuses on increasing the kernel pipeline employing loop unrolling (*#pragma unroll 2*) in the loop of the kernels code (when present and in the most internal one in case of nested loops). Increasing the number of computing units also extends HW utilization and memory contention (INTEL, 2020c). Loop unrolling replicates its body multiple times allowing it to reduce its trip count but also raising the HW resources (INTEL, 2020c).

Regarding the static and dynamic power of the CPU, since the first Case Study simulates the execution of 15 applications together, we estimate the CPU power in the following way. For each application a and power type pt (static or dynamic), we obtain the weighted power P_{pt} using the previously calculated power of each application ($Power_{a,pt}$) times the execution time of the application ($ExecTime_a$) during

Table 10 – HW Tasks' Latencies, Powers, and Partial Bitstreams - Two Compute Units

ID	Appl.	Freq. (MHz) (fmax)	Latency (us)	Latency (cycles)	Core Dynamic Power	Core Static Power	Core ALUTs	Core FFs	Core RAMs	Core DSPs	Partial Bitstream
1	2DCONV	340.28	24652.57	8388721	5.6180	4.2623	12529	33211	180	27	11023928
2	3DCONV	268.23	122.80	32939	5.7762	4.3577	19005	70589	354	57	11720396
3	2MM	284.72	30044508.44	8554272443	6.6277	4.5639	37288	70400	490	22	12126912
4	3MM	276.04	728443.88	201080864	8.2426	4.8393	55791	105319	726	33	13239924
5	ATAX	293.75	56842.69	16697540	5.8032	4.3415	25844	53588	370	10	11874256
6	BICG	278.65	59924.15	16697616	5.5110	4.2841	28406	60973	394	10	11875908
7	GEMM	305.56	219470.63	67060470	5.1610	4.2140	20831	41393	288	15	11172328
8	GESUMMV	298.30	28030.69	8361427	6.3002	4.4111	21167	56374	374	11	11732940
9	GRAMSCHM	296.88	14077.31	4179202	10.2987	5.2917	84437	123016	672	99	16279660
10	MVT	278.65	59955.14	16706251	6.1507	4.4047	43128	66036	374	22	12891816
11	SYR2K	266.20	16084574.30	4281773252	5.3001	4.2974	33195	61080	462	21	11319524
12	SYRK	318.75	1679493.42	535338529	5.4636	4.2598	20923	41307	290	15	11113488
13	CORR	285.15	4860531.87	1385980663	10.4023	5.3352	63018	145545	842	61	15752900
14	COVAR	292.61	2980841.40	872234842	8.0703	4.7920	37245	87512	488	28	13483516
15	FDTD-2D	302.08	20827.82	6291738	8.0762	4.8159	44421	94151	364	39	13598276

Table 11 – HW Tasks' Latencies, Powers, and Partial Bitstreams for Application 2MM (3) with FPGA Optimizations

Optimiz.	Imp. ID	Freq. (MHz) (fmax)	Latency (us)	Latency (cycles)	Core Dynamic Power	Core Static Power	Core ALUTs	Core FFs	Core RAMs	Core DSPs	Partial Bitstream
4 LU	4	248.26	17260797.25	4285232651	5.1298	4.2673	39670	68318	510	26	13177424
4 CU	5	270.83	15792522.41	4277136222	9.8094	5.2247	72668	137197	962	44	12249816
8 LU	6	236.11	9118911.59	2153076348	6.6940	4.6341	67198	110431	866	46	14631544
8 CU	7	224.54	9524346.15	2138568111	12.2300	6.0948	143458	270856	1906	88	12501400

the measurements, dividing it by the sum of the execution time of the applications ($\sum_{a=1}^{15} ExecTime_a$) – according Eq. 7. We apply the same estimation principle in the other case studies, only adjusting the applications according to the specific workload.

$$P_{pt} = (\sum_{a=1}^{15} Power_{a,pt} \times ExecTime_a) \div (\sum_{a=1}^{15} ExecTime_a) \quad (7)$$

To perform the FEHetSS's assessment in the Case Study related to *DSE Context*, Tab. 12 (settings 1 to 10) describes the used architecture settings, including information about the architectural scenario, type and number of PE, and the involved applications. In all settings, the frequency to A7, A15, and MALI are, respectively, 1.4GHz, 2.0GHz, and 0.6GHz. We explore the following scenarios in this Case Study:

- Homogeneous: only homogeneous CPU (A7 or A15).
- Asymmetric: distinct CPU processors making a big.LITTLE architecture (A7 and A15).
- Heterogeneous (GPU Accelerated): a big.LITTLE architecture augmented with a MALI GPU.
- Heterogeneous (GPU + FPGA Accelerated): a big.LITTLE architecture augmented with MALI GPU and Arria10GX FPGA, with the applications' kernels al-

Table 12 – Architecture Scenarios for Case Study #1 (settings 1-10))

ID	Scenario[, Acceleration, Optimization]	CPU	GPGPU	FPGA (# PRR)	Appl.
1	Homogeneous	8 A7	-	-	all
2	Homogeneous	8 A15	-	-	all
3	Homogeneous Asymmetric	4 A7 + 4 A15	-	-	all
4	Heterogeneous, GPU Accelerated	4 A7 + 4 A15	1 MALI	-	all
5	Heterogeneous, FPGA Accelerated	4 A7 + 4 A15	-	1 Arria10GX (1 PRR)*	all
6	Heterog., GPU + FPGA Accelerated	4 A7 + 4 A15	1 MALI	1 Arria10GX (1 PRR)*	all
7	Heterog., FPGA Accelerated	4 A7 + 4 A15	-	1 Arria10GX (2 PRRs)*	all
8	Heterog., FPGA Accelerated	4 A7 + 4 A15	-	1 Arria10GX (4 PRRs)*	all
9	Heterog., FPGA Accel., Optim. (2 comp. units)	4 A7 + 4 A15	-	1 Arria10GX (3† PRR)*	all
10	Heterog., FPGA Accel., Optim. (2 loop unrol.)	4 A7 + 4 A15	-	1 Arria10GX (4 PRRs)*	all

* Each application kernel operates at a different frequency (fmax).
† Regarding the use of four equally sized PRR, its size does not support each HW task implementation of the kernels.

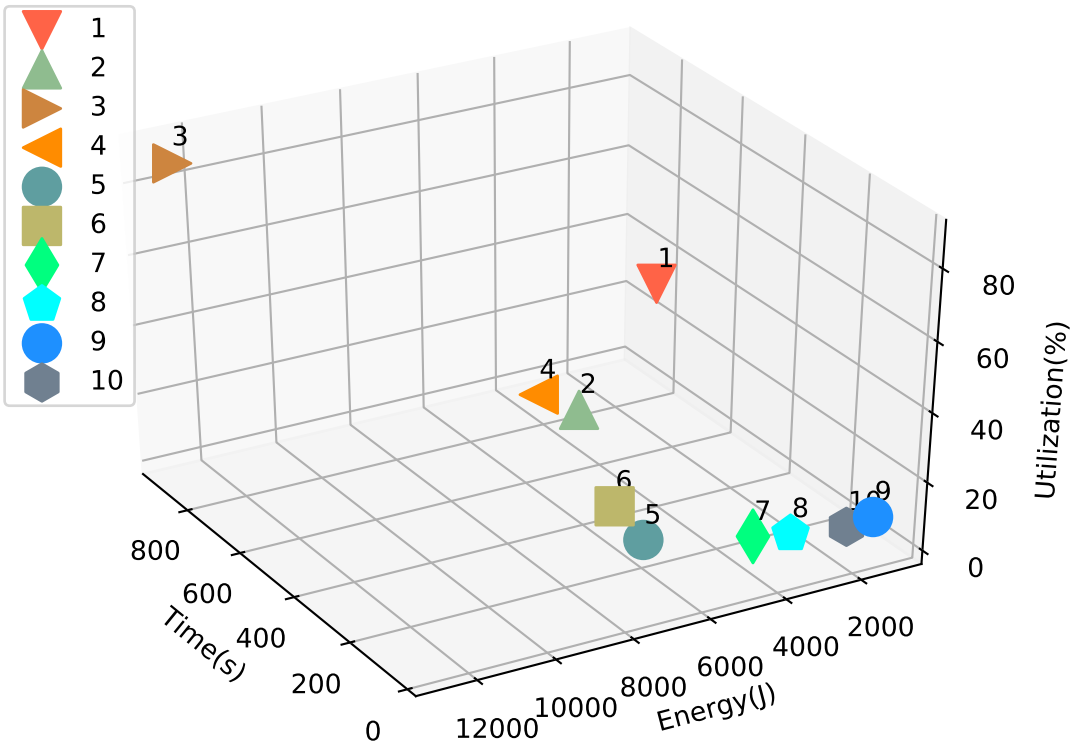


Figure 43 – Energy vs Time vs Utilization of Case Study #1 - DSE Settings (1-10 of Tab. 12).

ternating the mapping between these PE.

- Heterogeneous (FPGA Accelerated with multiples PRRs): a big.LITTLE architecture augmented with an FPGA device (Arria10GX), using a different number of PRRs.

5.2.1 Results of Case Study #1 – DSE of Heterogeneous Systems

This section presents the results of the Case Study regarding aspects of DSE (settings 1 to 10 of Tab. 12 – Sec. 5.2). Performing **DSE activities**, we can evaluate metrics of the selected DSE settings. Fig. 43 presents a 3D scatter plot showing the metrics Energy, Time, and Utilization regarding the execution of all applications – time measures the last finished application and energy is for the system, while utilization

is the average value between the PEs within the architecture. In Fig. 43, we can observe the heterogeneous architecture's advantages – especially when using FPGAs with multiples PRRs (settings 7 and 8) and employing optimized HW tasks (9 to 10). Multiple PRRs allow using the FPGA device as multiple PEs, each PRR executing a distinct HW task. Moreover, even with a large HW, heterogeneous FPGA settings (7–10) can achieve better energy efficiency. In these cases, even though the HW has a higher power dissipation, the HW tasks' performance within the PRRs allows a lower total energy consumption compared to other configurations.

The optimized configurations 9 and 10 provide the best trade-off for the metrics, showing the advantages of customized (and optimized) HW tasks to process the applications' kernels. Regarding settings 9 and 10, Fig. 44 shows a detailed plot describing the power consumption (Fig. 44 A and B) and HW utilization (Fig. 44 C and D) of each PE along time. These graphs show an important feature of the FEHetSS simulator – the ability to log metrics for each PE, allowing observe the system's behavior during the workload execution.We can notice the variation of power and utilization, identifying the points in time of PRRs reconfigurations. Fig. 45 presents a Power along with Time plot of setting 9 detailing the reconfiguration points of the PRRs (at top) and the completion of the application (at the bottom). For the reconfiguration points, we include a label in the format *r(App-PRR)* – e.g., *r(7-0)* indicates the configuration of the Application 7 HW task in the PRR 0. In the label of application completion, the plot shows the application ID. In this configuration (setting 9), we can note that the increase of computing units causes a higher power dissipation and HW utilization but compensates with lower execution time.

FEHetSS allows a rapid evaluation of VPs describing distinct architectures. The simulations of the case study #1 settings were ≈ 24 to ≈ 166 seconds long according to Fig. 46. The longer ones have associated with the "time worst" configurations since the simulation in these cases takes longer times. The preparation of the FEHetSS inputs demands applying the presented flow and some supporting tools. However, the possibility of quickly assessing design points from a system's point of view is an essential feature, even more during the initial phases of a design.

We model the applications' kernels as a sole HW task (Fig. 35 in Sec. 4.3.2 – task *t07-FPGA*), even if the implementation uses more than one OpenCL kernel – many Polybench applications have more than one kernel according Tab. 5. These coarse HW tasks use more resources and diminish the feasible number of PRRs. However, if we model each of these (sub) kernels as an independent HW task, more PRRs concurrently in use will be possible, but paying the price with more reconfiguration points. It is a possibility to evaluate in future FEHetSS's experimentations.

Ultimately, we can answer the question "*Based on high-level models, can FEHetSS provide appropriate log metrics in heterogeneous architectures' evaluation during a*

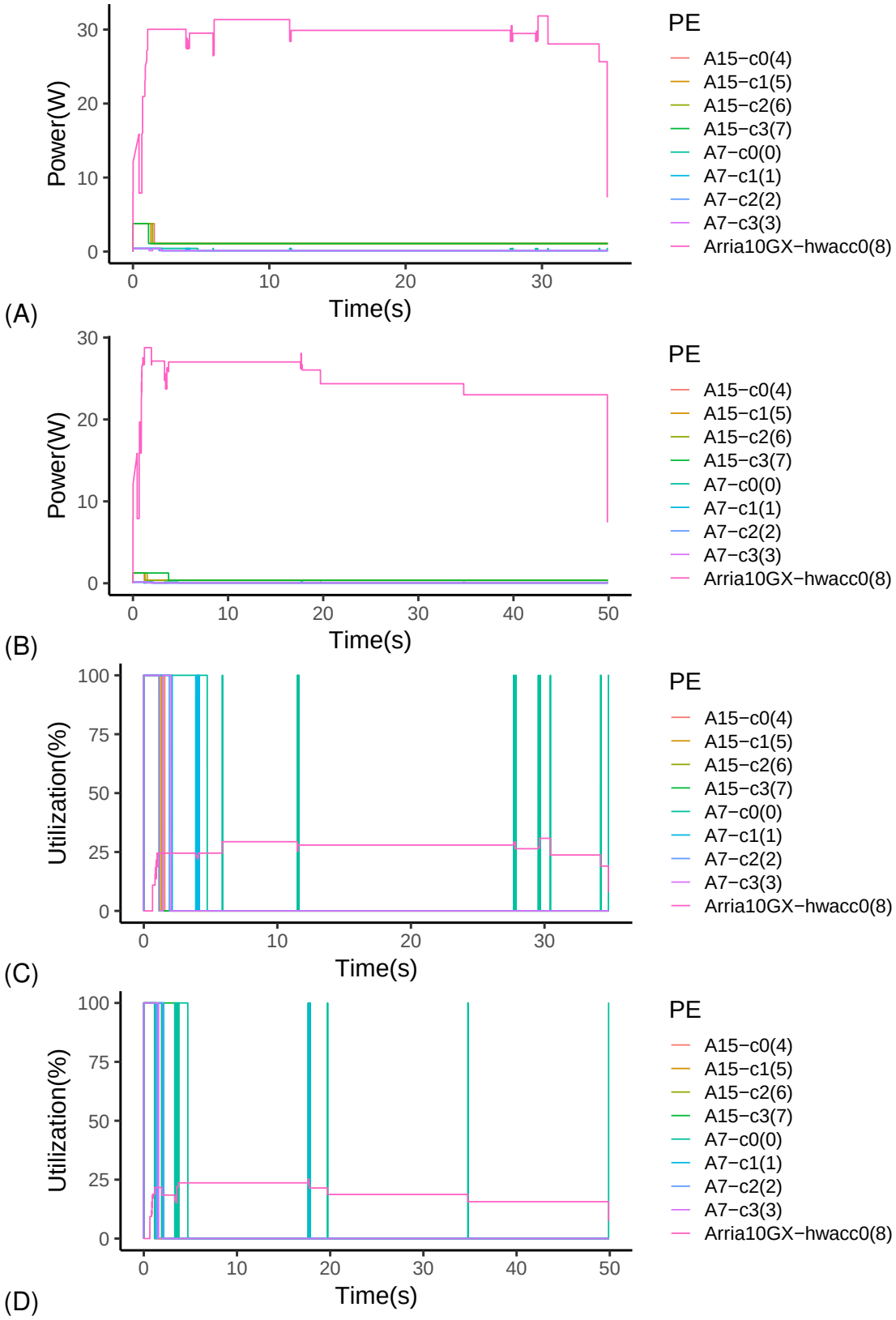


Figure 44 – Power Consumption (A and B) and HW Utilization (C and D) along time of each PE for settings 09 and 10, respectively (top to bottom).

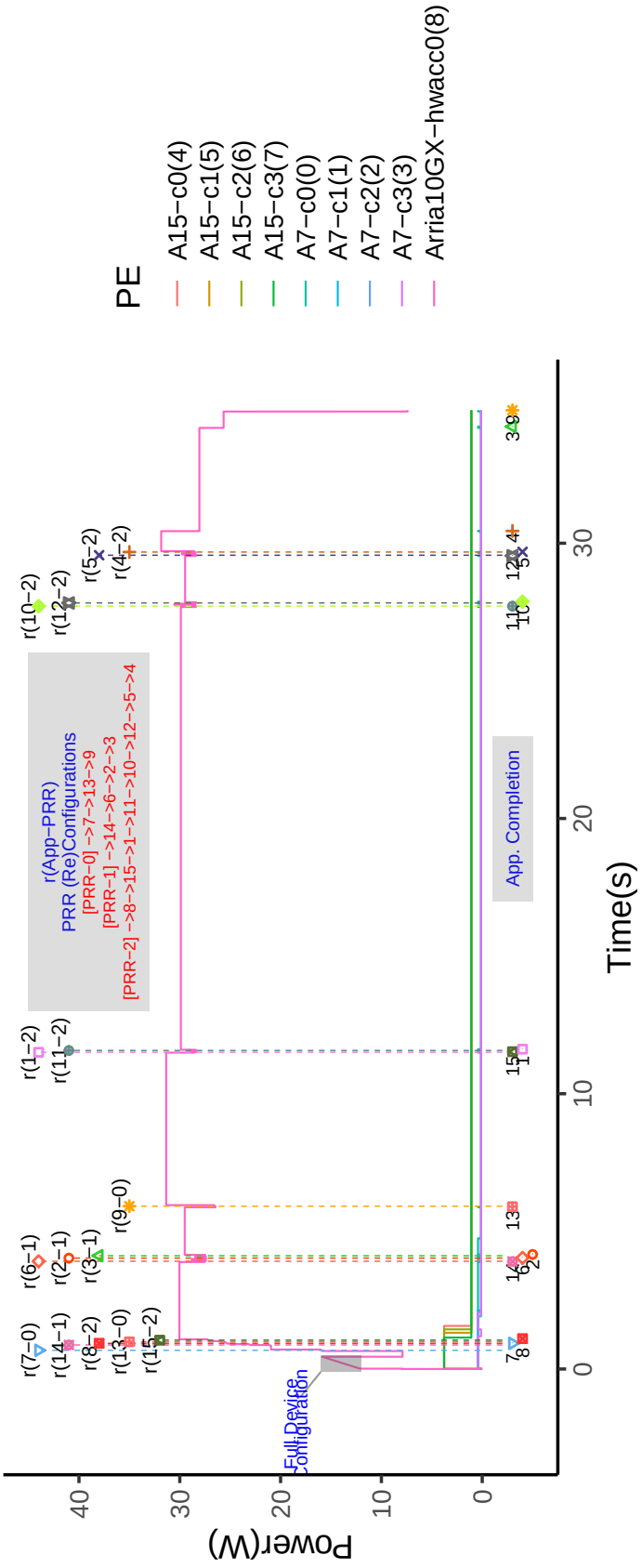


Figure 45 – Power Consumption of Setting 09 detailing the Reconfiguration Points (top) of each PRR of the Arria10GX Device and the Applications Completions (bottom).

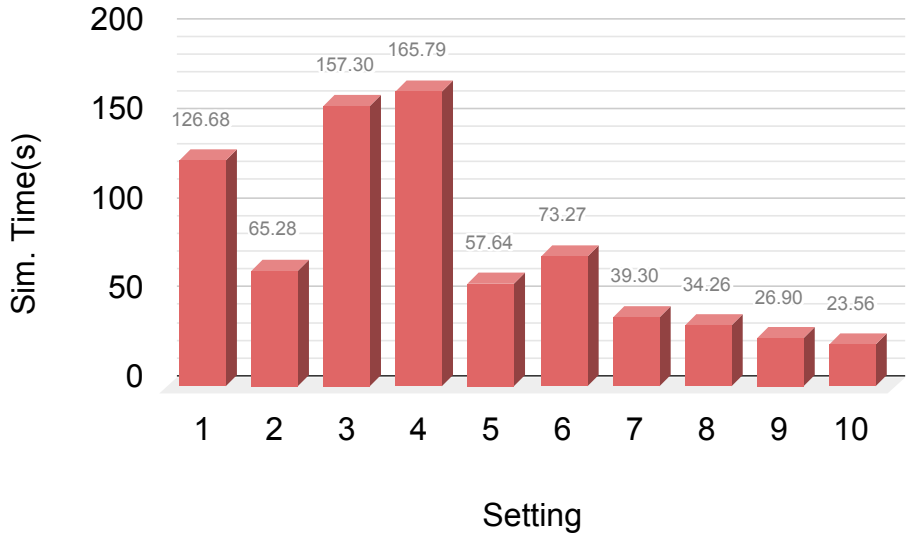


Figure 46 – Simulation Times for Settings 1 to 10.

Table 13 – Architecture Scenarios for Case Study #2 (settings 11-16)

ID	Scenario[, Acceleration, Optimization]	CPU	GPGPU	FPGA (# PRR)	Appl.
11	Heterog., FPGA Accel., Optim. (2 comp. units)	1 A15	-	1 Arria10GX (1 PRR)*	2MM
12	Heterog., FPGA Accel., Optim. (4 comp. units)	1 A15	-	1 Arria10GX (1 PRR)*	2MM
13	Heterog., FPGA Accel., Optim. (8 comp. units)	1 A15	-	1 Arria10GX (1 PRR)*	2MM
14	Heterog., FPGA Accel., Optim. (2 loop unrol.)	1 A15	-	1 Arria10GX (1 PRR)*	2MM
15	Heterog., FPGA Accel., Optim. (4 loop unrol.)	1 A15	-	1 Arria10GX (1 PRR)*	2MM
16	Heterog., FPGA Accel., Optim. (8 loop unrol.)	1 A15	-	1 Arria10GX (1 PRR)*	2MM

* Each application kernel operates at a different frequency (fmax).

(manual) DSE?"

The answer is *Yes*. Based on the presented methodological steps, we generate VPs models for the case study settings. Considering the (manual) DSE outputs, especially the metrics for time, energy, and utilization, the results provide a feasible way to evaluate design points basing early design decisions. The case study contains settings featuring HW optimizations and considering heterogeneous PEs to allocate in the kernel's execution. The custom settings using optimized HW tasks show the benefits of employing HW accelerators. Also, enhanced by the reconfigurable characteristics of FPGA devices. The total time of simulation takes almost 13 minutes for all ten configurations (if sequentially simulated).

5.3 Case Study #2 – Evaluating HW Design Alternatives

The second Case Study uses a single application – 3 (2MM). According to settings 11 to 16 of Tab. 13, the scenarios included in the architecture one CPU (A15@2.0GHz) and one FPGA device (Arria10GX1150 with one PRR). Moreover, the implementation of 2MM application extends the use of optimization statements including multiple compute units (2, 4 and 8 compute units) for the kernel execution, and loop unrolling (2, 4, and 8 as the unrolling factors) in the kernels' loop.

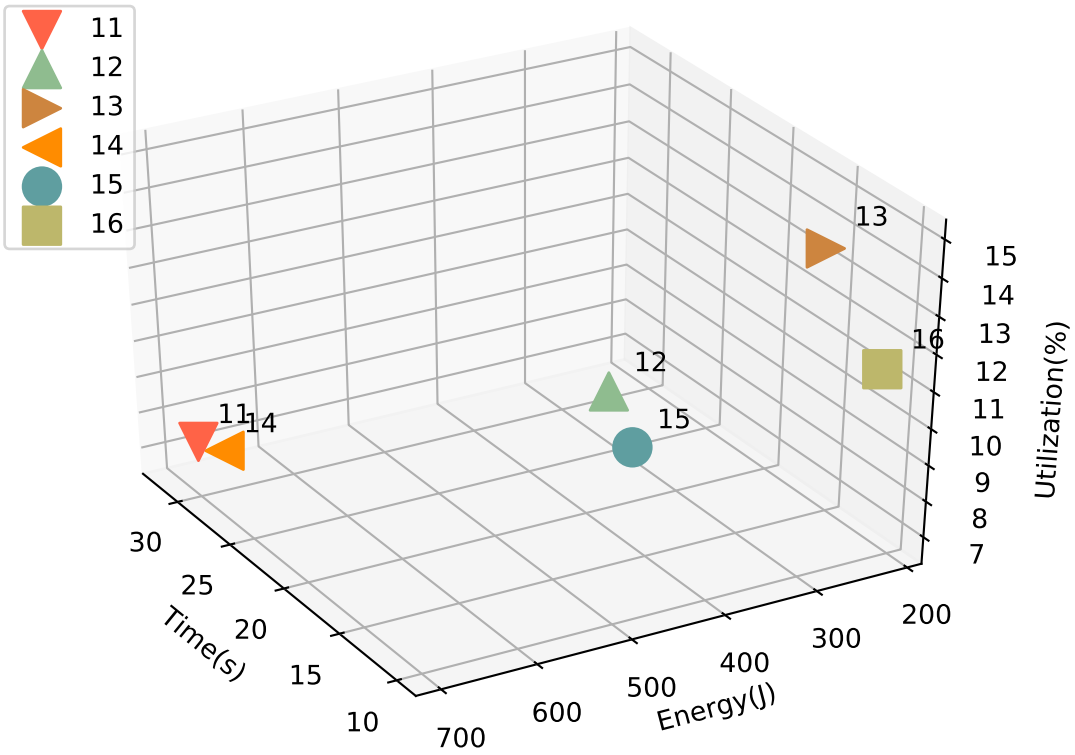


Figure 47 – Energy vs Time vs Utilization of HW Evaluation Settings (11-16 of Tab. 13).

Also aiming to evaluate the HW design alternatives, we planned another experiment with the 2MM (3) application. We prepare an exhaustive search regarding the different HW task implementations modeled for this application (IDs 1 to 7 in Tab. 7) and the available specific devices of Arria10GX family (IDs 1 to 9 in Tab. 6). The architecture settings include one CPU (A15@2.0GHz) and one FPGA device with one PRR. Thus, we define VPs for all possible combinations of the HW task implementations and the specific devices, totaling 63 settings. However, only 44 configurations are feasible for 2MM since we have some ”small” devices and ”big” implementations that do not match the respective HW resources.

5.3.1 Results of Case Study #2 – HW Designs Evaluation

This section presents the results of the Case Study #2 regarding aspects of HW design evaluations (settings 11 to 16 in Tab. 13) and the exhaustive search considering all the available HW tasks implementations and the specific devices of the Arria10GX FPGA family.

In the first part of Case Study #2, FEHetSS allows the evaluation of two optimization alternatives: (*i*) the increase of the parallelism using multiple compute units, and (*ii*) the augmenting of pipeline stages, decreasing the iterations through loop unrolling. Fig. 47 shows a scatter plot with Energy, Time, and Utilization for each HW design evaluation setting (11 to 16 in Tab. 13). Fig. 48 and 49 show the power consumption and utiliza-

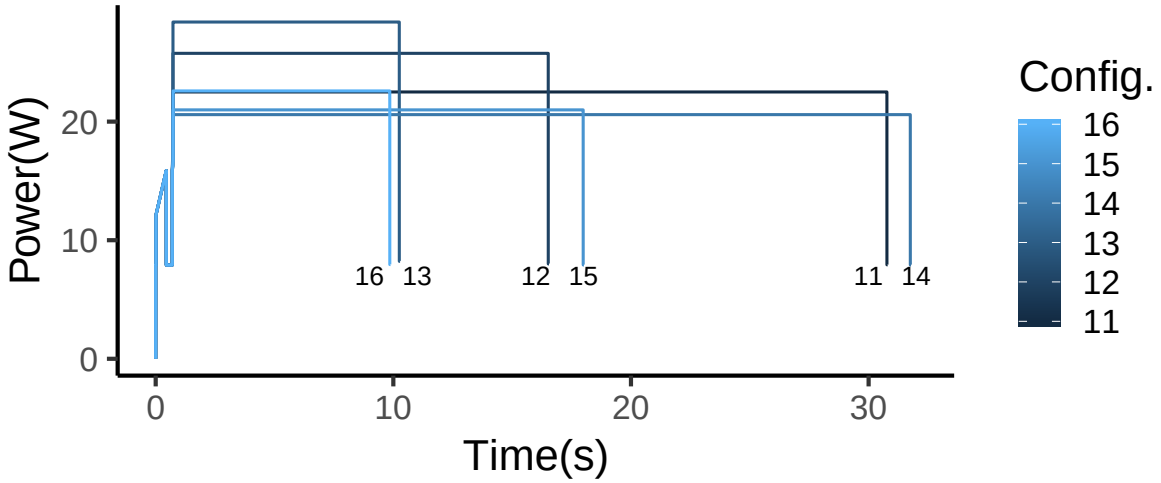


Figure 48 – FPGA Power Consumption along Time (settings 11 to 16 of Tab. 13).

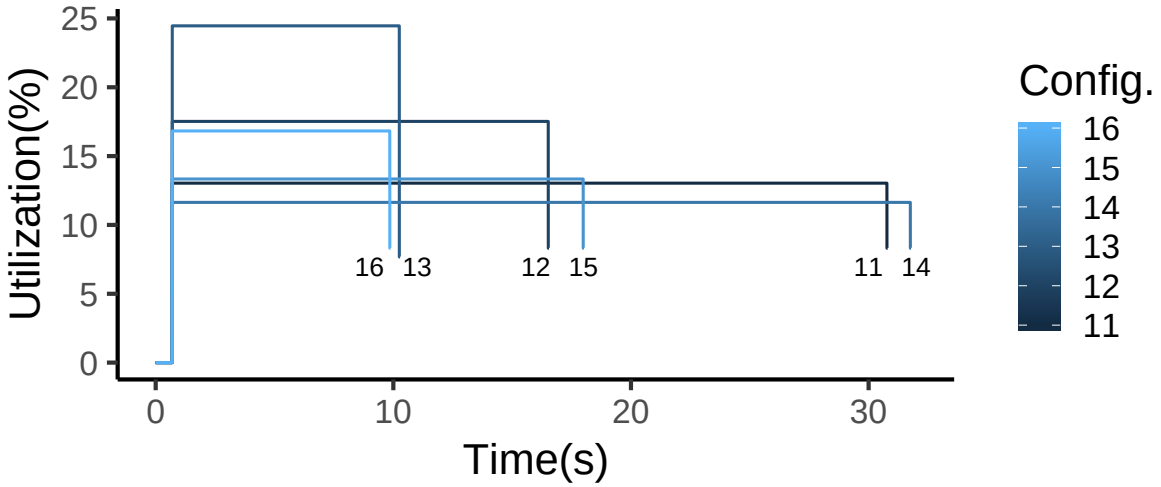


Figure 49 – FPGA HW Utilization along Time (settings 11 to 16).

tion, respectively, along time of the FPGA device for settings 11 to 16. We can observe the higher HW utilization in settings 12, 13, and 16 since they employ multiple computing units (4 and 8, respectively) and loop unrolling (setting 16 with 8-loop unrolling). Loop unrolling causes a smaller increase in HW resources. The variations of multiple compute units and loop unrolling influence power dissipation, but it is less intense with the latter. Further, the lower execution time and power consumption of setting 16 (with an eight loop unrolling factor) provide a better trade-off, providing the lowest energy consumption. For the 2MM (3) application, loop unrolling (with unrolling factor equal to 8) appears to be the more performance efficient optimization, overcoming the use of the same number of computing units. A designer can use FEHetSS to enable early exploration and evaluation of HW component design alternatives, including FPGA optimizations assessment. Fig. 50 shows the simulation times for the configurations 11 to 16.

Answering the question "*Can FEHetSS be used to assess different HW task imple-*

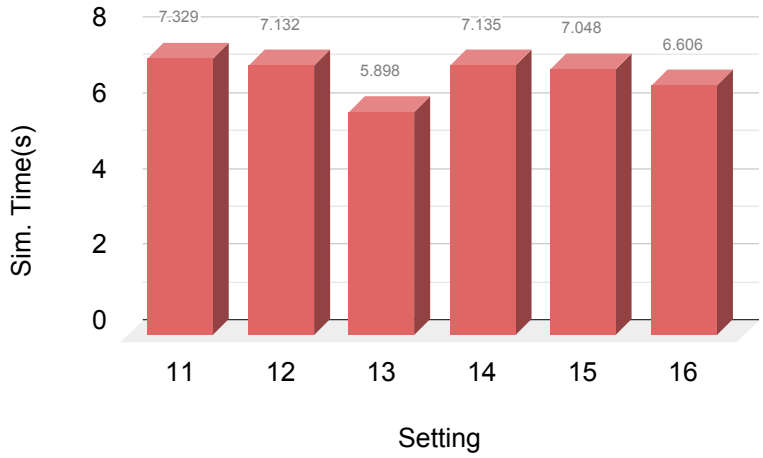


Figure 50 – Simulation Times for Settings 11 to 16.

mentations based on high-level models describing the tasks themselves and the HW accelerator device (FPGA)?", we believe the answer is *Yes*. When modeling optimizations, like loop unrolling and multiple computing units, each HW task implementation demands different HW resource amounts as power dissipation (and, consequently, energy consumption). Also, the task latency influences the applications' execution time. Thus, these aspects characterize an optimization problem since we need to find a trade-off considering time, energy, and HW utilization by varying the PRM implementation. Through OpenCL pragmas, we indicate to the HLS tools which optimization applies. Regarding FEHetSS models, the optimizations change the models' annotations in terms of kernel latency, core logic power, and HW resources demanding. Thus, a simple modification in the VP models allows FEHetSS to produce a valuable system design outcome.

Regarding scenario 2 of Case Study #2, we complement the first scenario by performing an exhaustive search of VPs (solutions) using the 2MM (3) application as the system's workload and evaluating the combinations of the available FPGA-related models in the VP repository, specifically the HW tasks implementations and the specific devices of the Arria10GX device family. Fig. 51 presents the 44 evaluated solutions and its Pareto front (PF). For each solution in the PF, we identify the respective pair {specific device, HW task implementation}. It is worth to note that the implementation six (6) – using eight-loop unrolling (Tab. 7) – appears five times in the 10-points PF, being accompanied by the devices 1 to 5 (Tab. 6). We can highlight the pairs {5, 6}, {4, 6}, and {3, 6}. These settings allow a finer tune between the eight-loop unrolling implementation (6) and devices five to three (5, 4, and 3), providing a lower energy consumption. The HW task implementation (6) can only match with the devices one (1) to five (5) since the smaller devices (6-9) cannot accommodate the respective modules (the HW task implementations). The smallest device (9) appears in the PF but only supports the simpler HW task – the original 2MM (ID 1 in Tab. 7).

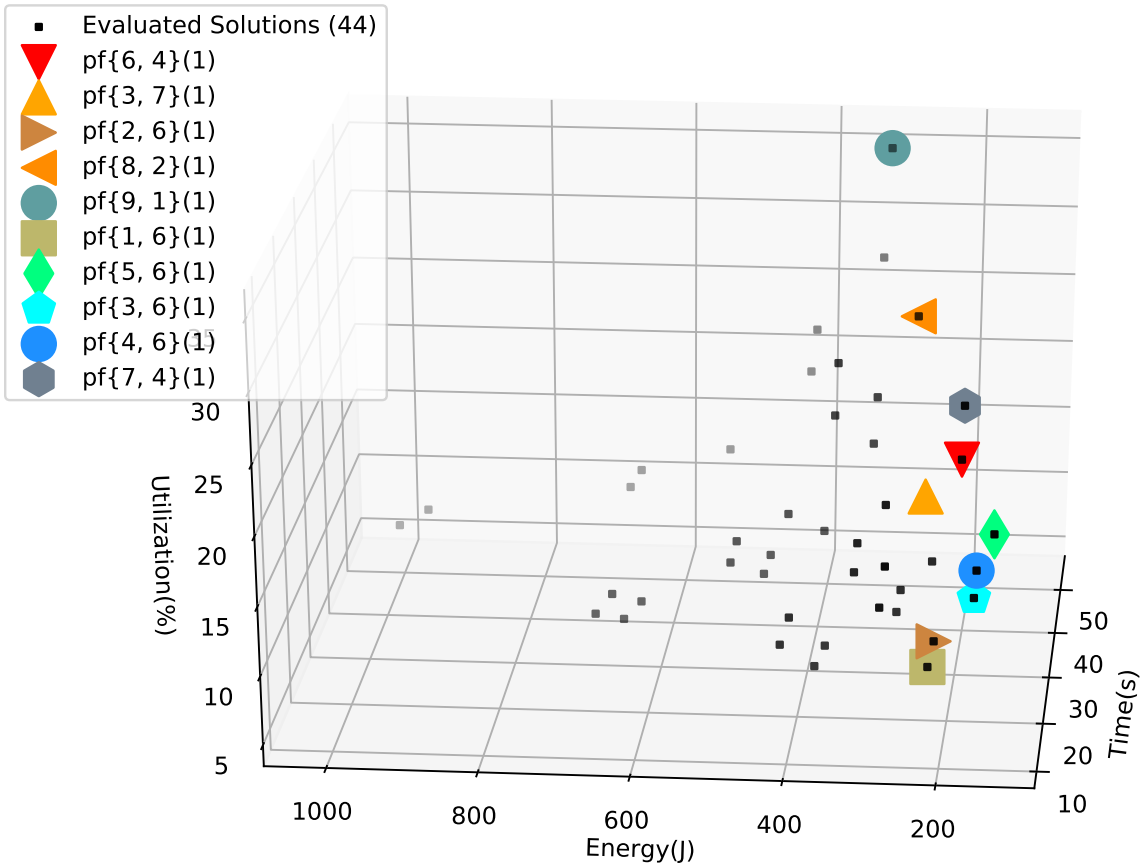


Figure 51 – Exhaustive Search Regarding the Specific Device (D) and the HW Task Implementation (I) for Application 2MM (3) [pf{D, I} – pf: Pareto Front]

We advocate the answer for the question:

"*Can FEHetSS be used to assess different HW task implementations based on high-level models describing the tasks themselves and the HW accelerator devices (FPGAs) with diversified resources?*" is also *Yes*.

In this case study scenario, we employ all Arria10GX devices (according to Tab. 6), combining them with the 2MM (Appl. 3) HW task implementations (Tab. 7) totaling 63 combinations. Through the search, some settings result in errors. It happens due to the non-matching between the available HW resources in a PRR and the respective PRM. In the end, forty-four combinations produce suitable outcomes. Utilizing the created models, we use FEHetSS to evaluate HW designs with different optimizations for an application's kernel. Besides, VPs simulations with FEHetSS allow the assessment of the device's "size" to employ in a system design.

5.4 Chapter Summary

This chapter has presented the first two experiments employing the modeling methodology and the FEHetSS simulator, organizing them according to the Case Study and Scenarios. Case Study #1 has promoted a manual DSE using different archi-

tecture types in ten settings with a mix of processing elements employed to process the applications' kernels. Scenario one of Case Study #2 has experimented with the FEHetSS's capability in evaluating HW design implementations considering an FPGA device also employing distinct optimizations for the HW task. In scenario two, we have combined FPGA devices from the Arria10GX family and HW optimizations.

6 EVALUATING FEHETSS IN A HEURISTIC-BASED DSE ENVIRONMENT

This chapter presents the Case Studies #3 and #4, both aimed to answer the Questions described in Sec. 6.1. In Case Study #3 (Sec. 6.2), we employ FEHetSS to rapidly evaluates solutions in conducting two exhaustive searches considering specific design spaces. Regarding Case Study #4, Sec. 6.3 evaluates the performance of FEHetSS integrated into an Optimization Heuristic-based DSE environment designing some simulation scenarios for them. This Case Study aims to provide a proof of concept that employing System-level Simulators (like FEHetSS) acting as a design point evaluation tool is a feasible and worth choice.

6.1 Goals for Case Studies #3 and #4

Sections 5.1.1 and 5.1.2 has presented the workload applications and the generation of model artifacts available in the VP repository for setting up simulations with FEHetSS. Thus, Case Studies #3 and #4 use the repository in the generation of the solutions. In this section, we present the goals of the experiments also using the GQM approach (BASILI; CALDIERA; ROMBACH, 1994). Tab. 14 shows the Case Studies and its scenarios by depicting its `Goal`s, the `Question`s to be answered, and the `Metric`(s) for evaluation to be employed in each case. In the metrics column, we insert a plot type aimed to show the results.

In the simulations of the Case Studies #3 and #4, we use a desktop with six Intel Core i5-8400 CPUs @ 2.8GHz, 64 bits, 8GB memory, running Linux Ubuntu 18.04.5 LTS Operating System. Whenever possible, we use several FEHetSS threads according to the number of available processing cores.

6.2 Case Study #3 – Using FEHetSS in Exhaustive Search

FEHetSS is a System-level Simulator that provides a rapid simulation of VPs. Aiming to show this feature, we planned exhaustive searches to perform with some Poly-

Table 14 – Summary of Case Studies #3 and #4

Case Study	Scenario	Goal	Question	Metric(s)	Architecture Type	Workload Appl.	Planning Section
#3	1	Assess FEHetSS simulator performing Exhaustive Searches of VPs aiming to find the design space PF.	Can FEHetSS be used to perform Exhaustive Searches in a feasible time, providing suitable Metrics to identify the design space's PF?	- System's Time, Energy, and Utilization. Graphs: 3D Scatter plot with Pareto Front regarding Appl. and its Kernel(s) Mappings.	Heterogeneous (GPU and FPGA)	4, 7, and 12	6.2
#3	2	Assess FEHetSS simulator performing Exhaustive Searches of VPs aiming to find the design space PF.	Can FEHetSS be used to perform Exhaustive Searches in a feasible time, providing suitable Metrics to identify the design space's PF?	- System's Time, Energy, and Utilization. Graphs: 3D Scatter plot with Pareto Front regarding Appl. and its Kernel(s) Mappings.	Heterogeneous (GPU and FPGA)	3, 14, and 15	6.2
#4	1	Evaluate FEHetSS integrated into a Heuristic-based DSE Environment acting as a design point evaluator. A set of DSE parameters configures the design exploration.	Can a designer employs FEHetSS as a solutions' evaluator into a Heuristic-based DSE Environment that produces good PF approximations of the design space?	- System's Time, Energy, and Utilization, and DSE Performance Indicators (HV and HR). Graphs: 3D Scatter plot with Pareto Front regarding Appl. and its Kernel(s) Mappings, and Boxplots for HV and HR	Heterogeneous (GPU and FPGA)	4, 7, and 12	6.3
#4	2	Evaluate FEHetSS integrated into a Heuristic-based DSE Environment acting as a design point evaluator. A set of DSE parameters configures the design exploration.	Can a designer employs FEHetSS as a solutions' evaluator into a Heuristic-based DSE Environment that produces good PF approximations of the design space?	- System's Time, Energy, and Utilization, and DSE Performance Indicators (HV and HR). Graphs: 3D Scatter plot with Pareto Front regarding Appl. and its Kernel(s) Mappings, and Boxplots for HV and HR	Heterogeneous (GPU and FPGA)	3, 14, and 15	6.3
#4	3	Evaluate FEHetSS integrated into a Heuristic-based DSE Environment acting as a design point evaluator. A set of DSE parameters configures the design exploration.	Can FEHetSS be integrated into a Heuristic-based DSE Environment providing diversified quality solutions? Also, demonstrating the exploration of the architecture features during the solutions quest?	- System's Time, Energy, and Utilization. Graphs: 3D Scatter plot with Pareto Front regarding Architecture Features.	Heterogeneous (GPU and FPGA)	1, 2, 5, 6, 8, and 10	6.3

bench applications. First, we define the VP's workload to initiate the applications 4 (3MM), 7 (GEMM), and 12 (SYRK) at the time zero. According to (COSTA et al., 2019), GPU is suitable to execute the kernel of the application 4 (3MM), as well as CPU has a better performance with 12 (SYRK), regardless of both PEs providing good performance with 7 (GEMM). Thus, we consider these applications to be from different profiles. Moreover, we reduce the design space restricting the architecture characteristics in the experiment scenario, as shown in the following items.

- 3 Applications from Polybench
  - 3MM (4), GEMM (7), and SYRK (12) - Tab. 5
- Architecture including:
  - A7 and A15 CPUs, Mali-T628 GPU, and Arria10GX FPGA as PE
- CPU clusters with 2 or 4 cores
- Frequencies range:
  - 600-1400 for A7 CPU in 200MHz steps
  - 600-2000 for A15 CPU in 200MHz steps
  - 600MHz for MALI GPU
  - 100 MHz for Arria10GX FPGA (except for the PRR)
- 1, 2, or 3 PRRs in Arria10GX FPGA device

- Specific Devices from Arria10GX Family (matching with the HW tasks)
  - Arria10GX320, Arria10GX480, and Arria10GX570 (IDs 6, 5, and 4 in Tab. 6)
- 3 HW task implementations
  - (1) original, (2) with 2-loop unrolling, and (3) with 2 compute units (IDs according to Tab. 7)

As a second exhaustive search, we elect other three applications of Polybench, also considering as from different profiles (COSTA et al., 2019) but being applications with higher computational cost in terms of execution time. The applications are 2MM (3), COVAR (14), and FDTD-2D (15). The architecture may include the same types of PEs as the previous search. Besides the number of CPUs in the clusters, the frequencies range, the number of PRRs, and the HW task implementations maintain the same as the first exhaustive search. The employed devices are Arria10GX480, Arria10GX570, and Arria10GX660 (IDs 5, 4, and 3 in Tab. 6).

For both scenarios, we prepare automation scripts that initiate separate threads executing different VPs. It logs the results of each simulation. It is worth mentioning the architecture/workload models of the VP may do not match each other – e.g., may occur a situation where an HW task demands more resources than the available on a PRR into an equally PRR-partitioned FPGA device. In this situation, FEHetSS produces an error message and the searching script passes to the next VP evaluation.

6.2.1 Results of Case Study #3 – FEHetSS in Exhaustive Search

This section shows the results of Case Study #3. The exhaustive searches consider 3MM (4), GEMM (7), and SYRK (12) applications in the first scenario. After, the 2MM (3), COVAR (14), and FDTD-2D (15) applications are the workload.

Fig. 52 (A) presents a scatter plot of the first space exploration using three dimensions – Time, Energy, and Utilization – it is worth to recall we desire solutions that minimize Time and Energy, maximizing Utilization. Also, Fig. 52 (B) shows only the Pareto Front (PF) solutions (VPs) of the evaluated Design Space. We also indicate the VPs profile presenting the mapping of the applications' kernel(s) using the following IDs: (1) CPU; (2) GPU; and (3) FPGA. For each profile, we present its number of solutions. In a similar way, Fig. 53 (A) presents a scatter plot of the second design space evaluation, also detailing the PF in Fig. 53 (B). We write a Python code, based on an idea presented in (SHIELDS, 2015), to extract the PF from the simulations logs.

Analysing Fig. 52, we notice the high number of PF solutions in the { (4,2) (7, 3) (12, 1) } profile. This indicates its mappings providing a good trade-off for the applications 4, 7, and 12. It also corroborates, in part, with the claims of (COSTA et al., 2019) indicating GPU to process the kernel(s) of application 4 and CPU to application 12.

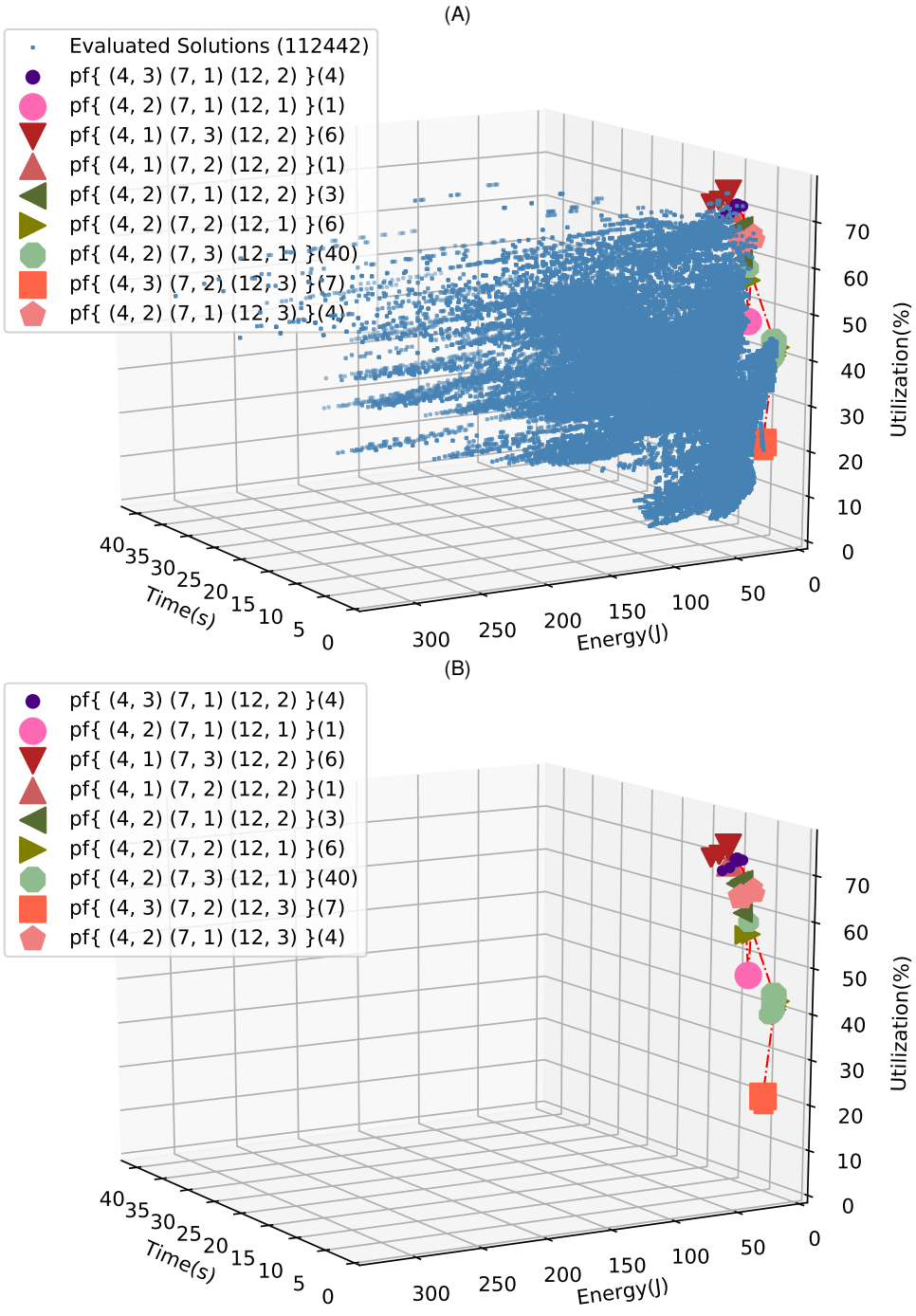


Figure 52 – Case Study #3: (A) Exhaustive Search for Appl. 4, 7, and 12. (B) Only its Pareto Front (PF). {(Appl. ID, Map. PE), ...}(# Solutions). PE - 1: CPU; 2: GPU; 3: FPGA.

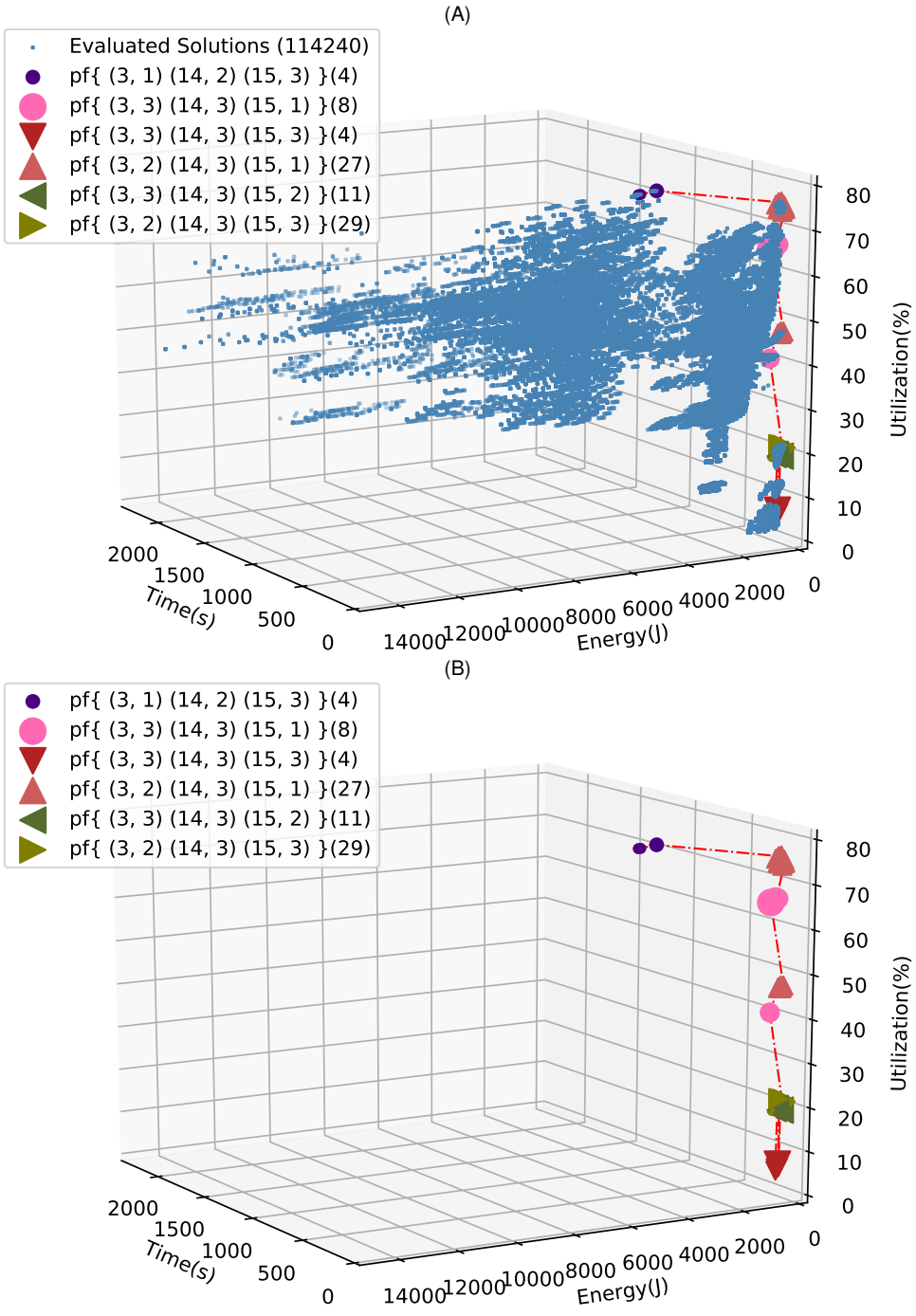


Figure 53 – Case Study #3: (A) Exhaustive Search for Appl. 3, 14, and 15. (B) Only its Pareto Front (PF). {(Appl. ID, Map. PE), ...}(# Solutions). PE - 1: CPU; 2: GPU; 3: FPGA.

The kernel(s) of application 7 better match with FPGA, since $\frac{46}{72}$ (63.88855%) of the PF solutions use this mapping, against $\frac{11}{72}$ (15.2771%) for applications 4 and 12 individually.

On the other hand, Fig. 53 presents fewer profiles in the PF solutions, but maintaining the majority ($\frac{56}{83}$, $\approx$ 67.46979%) in two profiles – pf{(3, 2) (14, 3) (15, 1)} and pf{(3, 2) (14, 3) (15, 3)} – where applications 3 and 14 show better results with the mappings to PE 2 (GPU) and 3 (FPGA), respectively. Moreover, we can perceive that FDTD-2D (15) application, specifically its kernel(s), enables a good performance when executed in any of the three evaluated PE (CPU(1), GPU(2), or FPGA(3)).

Usually, an exhaustive search is not an option when dealing with HW/SW systems due to the design space size. However, if a designer can diminish that space by restricting the search parameters, FEHetSS allows a rapid evaluation. In scenario 1, the search demanded a little over a day to complete (25.51 h). In the second situation, which contains time demanding applications, the demand took almost 15 days (14.72 days). In both scenarios, regarding architecture parameters, we have the following items: two values for the number of CPU cores; five and eight values in the A7 and A15 frequency ranges, respectively; two CPU models (A7 and A15); three values for PRR; three different devices; and three HW task implementations. About the applications/mappings, we have three possible kernel mappings for three different applications. Therefore, the number of solutions to evaluate in both exhaustive searches is $(2 \times 5 \times 8 \times 2 \times 3 \times 3 \times 3) \times (3^3) = 116640$. We recall that several solutions are not feasible due to incompatibility between the HW task implementation (PRM) and the PRR. The available HW resources on each PRR vary according to their number (with more PRRs, fewer HW resources are available). Thus, the number of feasible (evaluated) solutions were 112442 and 114240 for scenarios 1 and 2, respectively.

Aiming to answer the question "*Can FEHetSS be used to perform Exhaustive Searches in a feasible time, providing Metrics to identify the design space's PF?*", we evaluate two scenarios related to exhaustive searches. In both cases, FEHetSS evaluates more than one hundred thousand settings. The first considers a light-demanding application set, while the second includes applications that take longer simulation times. Even with reduced design space, the second exploration took several days to conclude. Both scenarios simulate the whole application. One way to shrink the exploration time is to model only the applications' kernels, discarding most of the other tasks. Although abstracting away a significant part of some applications, such reduction could still provide a workload core's evaluation in a shorter time. Although the produced logs characterize the design spaces, the time demand exposes a necessary trade-off in apply exhaustive searches – simulation time versus optimal Pareto front identification. Thus, we answer the question as follows: *Yes, but it depends on the design space size, simulation demand, and outcome breadth. If the designer can make their evaluation based on "reduced" search parameters, the time to pay in simulation*

and searching may be worth.

6.3 Case Study #4 - Evaluating FEHetSS in an Heuristic-based DSE

We prepare a DSE environment including an optimization heuristic – Simulated Annealing (SA) (FRANZIN; STüTZLE, 2019) – and our system-level simulator – FEHetSS. SA guides the exploration of the design space, and FEHetSS evaluates the design points during DSE. A set of C++ functions implements the simulated annealing heuristic. It integrates FEHetSS in the heuristic strategy by initiating simulation threads that evaluate a given solution (a virtual platform). Figure 27, described in the beginning of Chapter 4, provides a representative view of the environment components and its interactions.

Based on the DSE parameters, the heuristic deals with the solutions, randomly creating and altering them, using models available in the Virtual Platform (VP) repository. The DSE parameters contain the alternatives or range of values of the VP elements. As for architecture, the parameters describe the feasible PE types, the number of PEs, the frequency range for each PE model, the number of PRRs (for HW accelerators – FPGA) to be used, the available specific devices (for FPGA), and the HW task implementations usable as HW tasks (FPGA). The parameters describe the workload defining each application (as an ID) and its arrival time in the simulation.

VP repository contains power models for the architecture elements (PE), sorted by the PE type (CPU, GPU, and FPGA), its respective model, the application, and the frequency range. Regarding the workload, the available application's models describe the performance annotations considering the different types of tasks (elaboration or main) existing in the applications and according to a frequency range. We can use the VP repository in several ways: manually, through an automation script, or within a DSE environment/heuristic. A collection of text files (in XML format) organized in folders according to the applications and processors (PE) make up the VP repository. On the other hand, a set of shell scripts allows creating a FEHetSS's input file that describes a VP – an XML file with a structure according to Fig. 37 (Chap. 4).

SA creates random settings as candidate solutions to evaluate during the DSE by using the FEHetSS simulator. According to the metrics provided by FEHetSS, the evaluation model component provides the solution evaluation. Using a Random Walk component, SA extracts the average evaluation differences ($avg\Delta$) between two consecutive solutions evaluating a fixed number of random solutions. This averaged value dictates the initial and final Temperature to use in the DSE processing. After the random walk, SA initiates its flow using stop and acceptance criteria to assess the DSE termination and acceptance of solutions. The Temperature parameter controls the

Table 15 – Architecture Types Description and Its Processing Elements

ARCH ID	Architecture Description	A7	A15	MALI	Arria10GX
0	Homogeneous (LITTLE - A7)	✓	✗	✗	✗
1	Homogeneous (big - A15)	✗	✓	✗	✗
2	Asymmetric (LITTLE, big)	✓	✓	✗	✗
3	Heterogeneous - GPU Accelerated (LITTLE, Mali)	✓	✗	✓	✗
4	Heterogeneous - GPU Accelerated (big, Mali)	✗	✓	✓	✗
5	Heterogeneous - GPU Accelerated (LITTLE, big, GPU)	✓	✓	✓	✗
6	Heterogeneous - FPGA Accelerated (LITTLE, FPGA)	✓	✗	✗	✓
7	Heterogeneous - FPGA Accelerated (big, FPGA)	✗	✓	✗	✓
8	Heterogeneous - FPGA Accelerated (LITTLE, big, FPGA)	✓	✓	✗	✓
9	Heterogeneous - FPGA+GPU Accelerated (LITTLE, GPU, FPGA)	✓	✗	✓	✓
10	Heterogeneous - FPGA+GPU Accelerated (big, GPU, FPGA)	✗	✓	✓	✓
11	Heterogeneous - FPGA+GPU Accelerated (LITTLE, big, GPU, FPGA)	✓	✓	✓	✓

DSE. SA's algorithm includes procedures for cooling, restarting, or increasing this parameter. The next section describes the Simulated Annealing (SA) heuristic integrated with a DSE environment.

6.3.1 Simulated Annealing as a DSE Heuristic

We adapt the component-based SA description presented by (FRANZIN; STüTZLE, 2019) in a DSE environment that includes the FEHetSS simulator as a solution evaluation component. The simulated annealing (SA) module separately executes the DSE for each type of architecture. Alg. 3 describes the SA heuristic used in the DSE environment. Alg. 4 describes the strategy to explore the neighborhood of an initial and random generated solution. Once a solution assessment is necessary, the SA fires threads from the FEHetSS simulator to perform such an evaluation.

In the algorithms, for the evaluation of a solution S, we use the magnitude of a 3D vector (Eq. 8) – from the origin (0, 0, 0) the lower the better – in a space with the dimensions $Time$, $Energy$, and $1.0 \div Utilization$ (inverse of the utilization) obtained with a $FEHetSS$ simulation. Thus, when comparing two solutions (VPs), the best is that with the lowest evaluation value. The purpose regarding $Time$ and $Energy$ is minimization. About $Utilization$, the goal is maximization intending to find solutions that use most of the architectural elements. We advocate this strategy aiming to minimize the system cost.

$$S.Evaluation = \sqrt{S.Time^2 + S.Energy^2 + (1 \div S.Utilization)^2} \tag{8}$$

The algorithms refer to an Architecture Type code (defined by an ID). It describes the PE models present in the solution (VP). This information indicates to the DSE the feasible elements to include in the VP. The VP repository contains models of the A7 (LITTLE) and A15 (big) CPUs, MALI-T628 GPU, and Arria10GX FPGA. Tab. 15 describes the Architecture Types indicating the PE Models included on each.

A set of parameters configures the DSE execution using the SA heuristic, according to Fig. 54. These parameters describe the range of possible values of each charac-

Algorithm 3 DSE using the Simulated Annealing (SA) Heuristic – based on the SA formulation presented on (FRANZIN; STüTZLE, 2019)

.

Require: Architecture Type ($arch$) - featured by its processing elements (PE)
Require: $current$, $next$, $best$ - solution $\langle description, results \rangle$

1: $\langle avg\Delta, current \rangle \leftarrow RandomWalk(rwSize, arch)$
▷ $rwSize$ is the size of the Random Walk
▷ Random Walk evaluates $rwSize$ solutions (with $FEHetSS$) and calculates the average *Delta* ($avg\Delta$) value between the $Evaluation$s of two subsequent solutions to use in the calculation of the Initial and Final $Temperature$s. Besides it returns the $best$ evaluated solution.
2: $best \leftarrow current$ ▷ Maintain the $best$ solution so far
3: $i \leftarrow 0$
4: $\langle InitialTemp, FinalTemp \rangle \leftarrow InitializeTemperature(avg\Delta)$
▷ Calculate the $Temperature$s – initial and final
▷ According to $avg\Delta$, calculates the necessary temperature to get a user-defined initial probability (0.9) to accept worsening solutions. Similarly, calculates the final temperature (probability of 0.001)
5: $T \leftarrow InitialTemp$
6: **while** $StoppingCriterion(T)$ is not met **do**
▷ Stops when the $Temperature\ T$ achieves $FinalTemp$
7: $i \leftarrow i + 1$
8: $next \leftarrow EvaluateNeighborhood(arch)$
▷ Creates a new solution, evaluates it, and its "near" neighbors returning to $next$ the best-evaluated solution.
▷ $EvaluateNeighborhood$ is described in Alg. 4
9: **if** $AcceptanceCriterion(next, current, T)$ **then**
▷ $\Delta = next.Evaluation - current.Evaluation$
▷ if $(\Delta < 0.0) \vee (e^{\frac{-\Delta}{T}} >= randomValue())$
10: $current \leftarrow next$
11: **if** $ImprovesOver(current, best)$ **then**
▷ if $current$ is better than $best$
12: $best \leftarrow current$ ▷ update $best$
13: **end if**
14: **end if**
15: **if** $TemperatureLength(T)$ is met **then** ▷ at least one accepted solution
16: $T \leftarrow CoolingScheme(T)$ ▷ $T = T \times 0.90$
17: **end if**
18: $T \leftarrow TemperatureRestart(T)$
▷ restart T if k (10) moves without an accepted solution
19: **end while**
20: return $best$(s)

Algorithm 4 EvaluateNeighborhood

Require: Architecture type ($arch$) - featured by its processing elements (PE)
Require: maximum $concomitantThreads$ for VP simulations using $FEHetSS$
1: $initSol \leftarrow NewCandidateSolution(arch)$ ▷ New random (initial) solution
2: $best \leftarrow EvaluateSolution(initSol)$
▷ Evaluate the received solution with $FEHetSS$ maintaining it as the $best$ solution so far.
3: $n \leftarrow 0$
4: $size \leftarrow NeighboursToEvaluate(arch, initSol)$ ▷ Identify the $changeable\ features$ (and its number) to modify in the initial solution. Changeable features are characteristics with more than one possible value in the VP description.
5: **while** $size$ neighbours is $not\ evaluated$ **do** ▷ Evaluates each one of the $size$ neighbours – according to the number of the $changeable\ features$
6: $nThreads \leftarrow$ suitable value according $concomitantThreads$ and $size$
7: **for** $a \leftarrow 1$ to $nThreads$ **do**
8: $Neighborhood_a \leftarrow Neighbour(initSol, arch, n + a)$
▷ creates a "near" neighbour (identified by $n + a$) changing one feature in $initSol$
9: $Neighborhood_a \leftarrow fork(EvaluateSolution(Neighborhood_a))$
▷ creates a thread for the $Neighborhood_a$ evaluation using $FEHetSS$
10: **end for**
11: $wait()$ ▷ wait the threads join
12: **for** $a \leftarrow 1$ to $nThreads$ **do**
13: **if** $ImprovesOver(Neighborhood_a, best)$ **then**
▷ Verifies if $Neighborhood_a$ is a better solution than $best$
14: $best \leftarrow Neighborhood_a$ ▷ maintain the $best$ solution
15: **end if**
16: **end for**
17: $n \leftarrow n + nThreads$
18: **end while**
19: return $best$ ▷ return the $best$ solution between the evaluated neighbours

teristic of the solutions. Based on the DSE parameters, SA can generate random solutions (Virtual Platforms - VPs) – each one with the structure described in Fig. 55, called VP description. A VP description indicates what PEs participate in the system's architecture and what applications are in the system's workload, also defining the mapping of the applications' tasks (including HW tasks to run in an FPGA device) in the architecture's PE. Fig. 55 also illustrates the changeable features of a solution. Alg. 4 uses these features to explore the (near) neighbors of a randomly generated solution – i.e., generates a new solution that is near from the original one, but with at least one different feature. A VP mainly contains the Architecture (ARCH) and the Workload (WRKLD) descriptions, also detailed in Figures 56 and 57, respectively.

Fig. 56 shows the alternatives and possible values of the architecture's PE characteristics. The PE features description includes its type, number, model, and frequency,

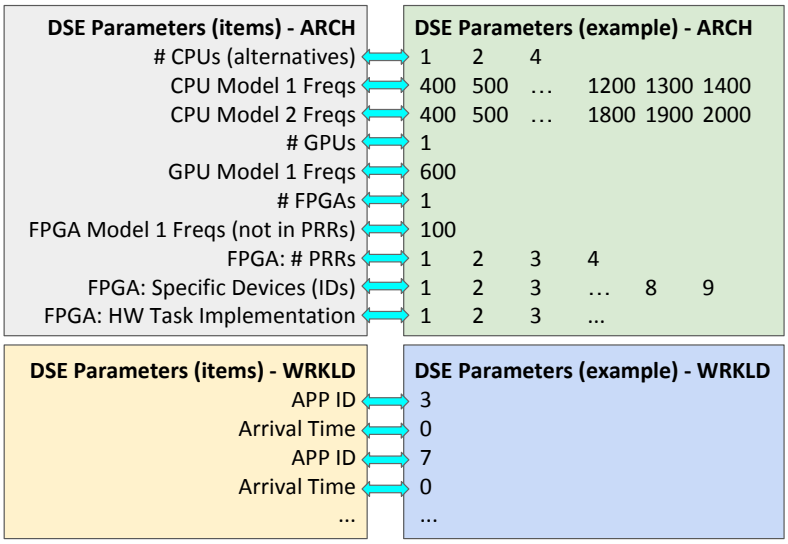


Figure 54 – Parameters for DSE Configuration

also describing the number of PRRs and specific devices in the case of FPGA. Besides, Fig. 57 illustrates the elements that model each application of the workload, indicating its ID and arrival time, beyond the PE type, its model, and frequency for the reference mapping of the main tasks, as well as for the elaboration tasks. For the elaboration tasks mapped to FPGA, the description indicates which HW task implementation to use. All these VP model elements are available in the VP repository presented in Sec. 4.1.

6.3.2 Heuristic-based DSE Experimentation

Integrated into a heuristic-based DSE environment, the experimentation with FE-HetSS follows the hypothesis that a System-level Simulator provides, in a feasible time, suitable metrics in the early phases of the system's design, even based on high-level models describing the VP. With this in mind, we prepare DSE executions considering some of the Polybench applications as workload. To allow the DSE's performance evaluation, we planned to employ the outputs of Case Study #3, presented in Sec. 6.2, as a basis for comparison between the DSE and Exhaustive Search simulations. As previously described in this section, the DSE Environment receives some parameters that direct the exploration. Thus, aiming to allow a fair comparison, we employ compatible parameters with the exhaustive searches presented in Sec. 6.2. We plan three experimentation scenarios using the DSE environment, as follows:

1. Applications 4 (3MM), 7 (GEMM), and 12 (SYRK) running in an architecture type 11 – Heterogeneous: FPGA+GPU Accelerated (LITTLE, big, GPU, FPGA) – since the time zero. The possible HW task implementations for the applications' kernel are 1, 2, and 3, according to Tab. 7. The CPU clusters may contain two or four cores using a sparse frequency range each (400-1400 MHz and

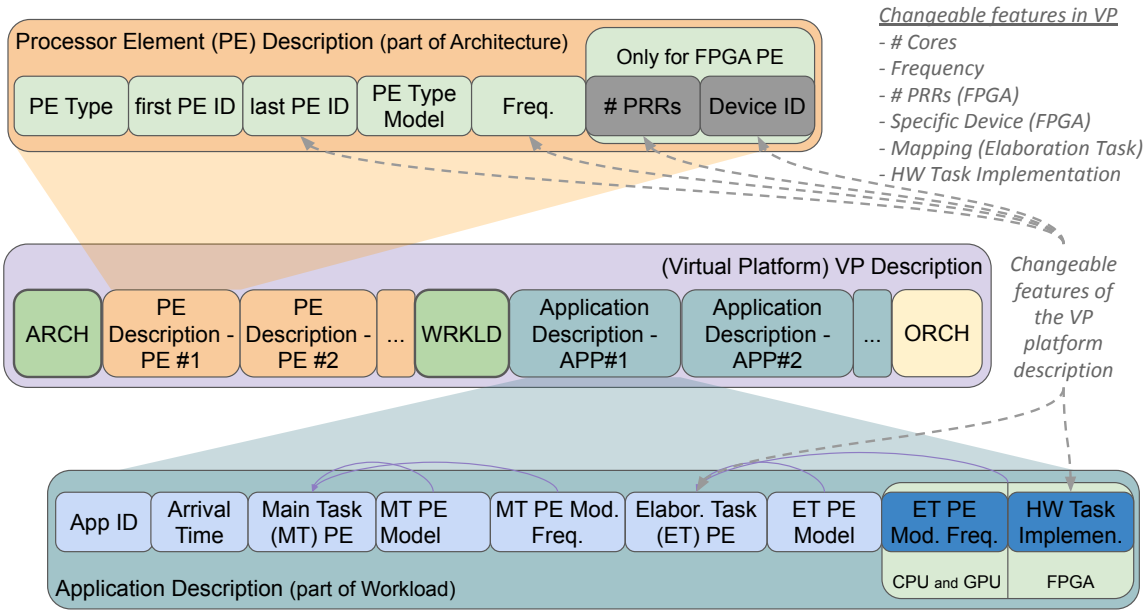


Figure 55 – Virtual Platform (VP) Description

400-2000MHz, in steps of 200 MHz, for A7 and A15 cores, respectively). The architecture may contain only one GPU (at 600MHz) and one FPGA (at 100MHz, except for PRR portions). The latter with 1, 2, or 3 PRRs. The specific FPGA devices are 4, 5, and 6 (according to Tab. 6). We run 30 times this scenario aiming to evaluate the DSE performance.

2. Applications 3 (2MM), 14 (COVAR), and 15 (FDTD-2D) initiating at time zero. The architecture features are the same of scenario 1, except for the FPGA devices – IDs 3, 4, and 5 according to Tab. 6. This scenario simulates 15 times in the DSE Environment.

3. Applications 1 (2DCONV), 2 (3DCONV), 5 (ATAX), 6 (BICG), 8 (GESUMMV), and 10 (MVT) starting at time zero. Tab. 16 shows the DSE parameters for this scenario.

To evaluate the DSE results, specifically the experimentation scenarios 1 and 2, we employ the output of the first and second experiments of Case Study #3 (Sec. 6.2), respectively. These outputs come from exhaustive searches and contain a Pareto Front (PF). Therefore, these PFs can represent the optimal solutions of a Design Space. It is suitable to employ in the DSE output's evaluation since we can extract the respective PF from the DSE outputs and use them as PF approximations. We implement a Python code, based on a sample code from a *stackoverflow* forum (SHIELDS, 2015), to extract the PF from the DSE logs.

Considering the PF and its approximations, we can use performance indicators to evaluate the DSE. One of these indicators is the Hypervolume (also known as *S* metric, hyper-area or Lebesgue measure), one of the most used and relevant measures

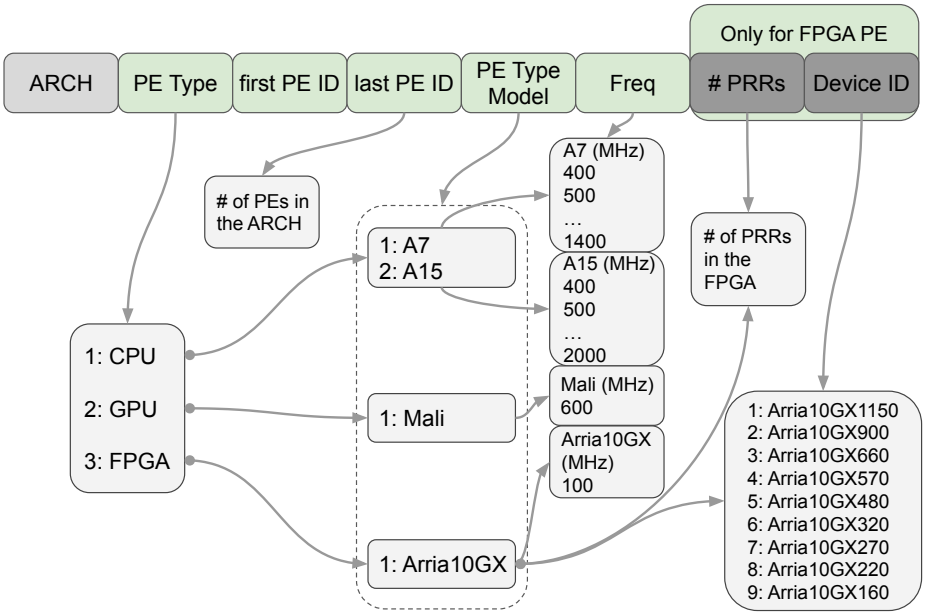


Figure 56 – Architecture Description Details

to evaluate Multi-Optimization Problems according to several authors (RIQUELME; VON LüCKEN; BARAN, 2015)(HALIM; ISMAIL; DAS, 2020)(AUDET et al., 2020). It is defined as a volume in the objective function space that covered by $p_i(i = 1, \ldots, N)$ of non-dominated set solutions (HALIM; ISMAIL; DAS, 2020). Hypervolume is a unary metric that measures the size of the objective space covered by an approximation set (RIQUELME; VON LüCKEN; BARAN, 2015) – a PF approximation. It is the volume of the space in the objective space dominated by the PF approximation A and delimited from above by a reference point $r \in \mathbf{R}^m$ such that for all $z \in A$, $z \prec r$ (AUDET et al., 2020) ($\prec$ means *dominates*). Hypervolume (HV) is given by Eq. 9 , where λ_m is the m-dimensional Lebesgue measure (AUDET et al., 2020). Fig. 58 illustrates the Hypervolume indicator in a two objective space (FONSECA; PAQUETE; LOPEZ-IBANEZ, 2006). A reference point (*r* in Fig. 58) must be used to calculate the covered space (RIQUELME; VON LüCKEN; BARAN, 2015).

$$HV(A, r) = \lambda_m(\bigcup_{z \in A}[z; r]) \tag{9}$$

We employ a HV implementation (WESSING, 2020), available in (RUDOLPH, 2020), to calculate the HV indicator. This code is a reimplementation of the HV calculation algorithm presented in (FONSECA; PAQUETE; LOPEZ-IBANEZ, 2006), being able to extract the HV from PF in design spaces of two or more dimensions.

Hyperarea ratio (HR) is another relevant indicator. If the PF is known (as the PF of an exhaustive search) and given a reference point r, the HR of an approximation A (as

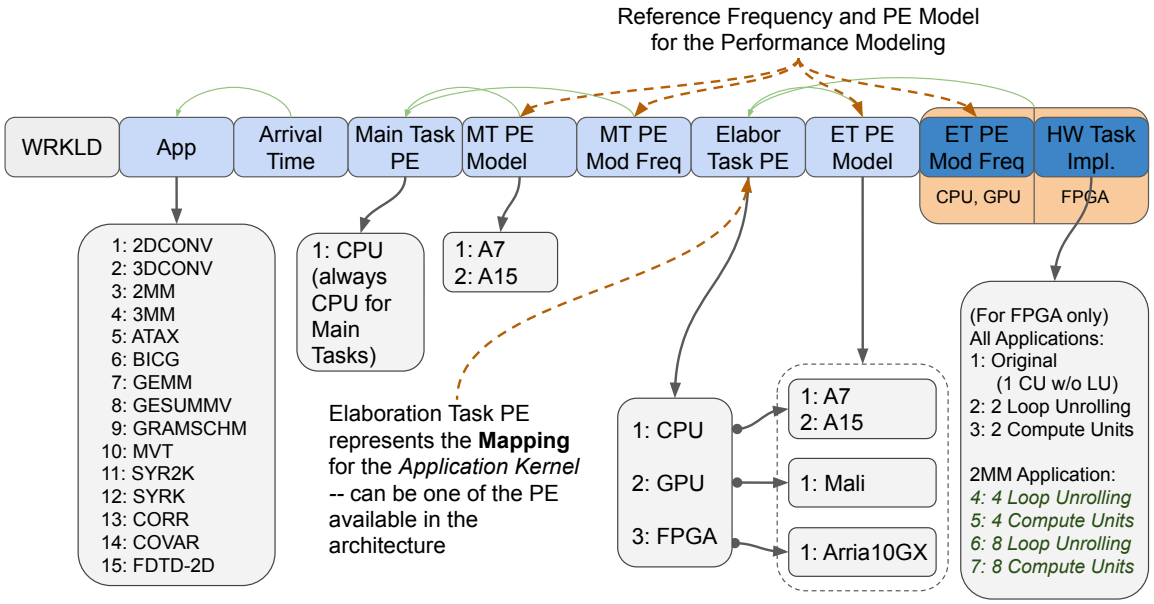


Figure 57 – Workload Description Details

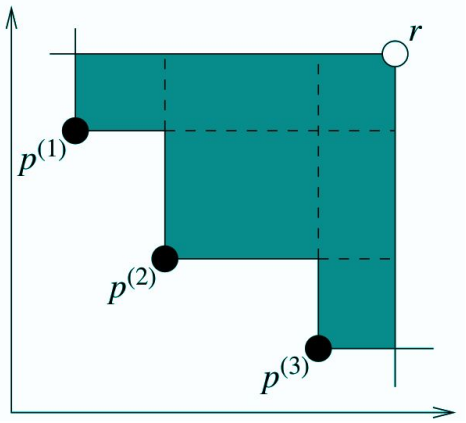


Figure 58 – Hypervolume Indicator in a Two-Objective Space (extracted from (FONSECA; PAQUETE; LOPEZ-IBANEZ, 2006))

the DSE output's PF) is given by Eq. 10.

$$HR(A, PF, r) = \frac{HV(A, r)}{HV(PF, r)} \tag{10}$$

The lower the ratio is (converges toward 1), the better the approximation is (AUDET et al., 2020). Both HV and HR are convergence and distribution indicators (AUDET et al., 2020).

Regarding the reference point r, we employ a simple strategy to calculate it. Considering all points of a PF in the exhaustive search, we calculate the respective Nadir point (WANG; HE; YAO, 2017). We can derive Nadir point using the extreme points of a non-dominated solution set (PF) considering the objective functions in a design space (WANG; HE; YAO, 2017). Fig. 59 presents an illustration of a Nadir Point in a two-dimensional space, additionally showing the *Ideal* and *Worst*

Table 16 – DSE Parameters for Experimentation Scenario 3

Feature	**Parameters**
Architecture Types	6 7 8 9 10 11 (according to Tab. 15)
# CPUs (alternatives for the number of CPU cores in a cluster)	2 3 4 5 6 7
CPU Model 1 Freqs (A7)	400 500 600 700 800 900 1000 1100 1200 1300 1400
CPU Model 2 Freqs (A15)	400 500 600 700 800 900 1000 1100 1200 1300 1400 1500 1600 1700 1800 1900 2000
# GPUs	1 2
GPU Model 1 Freqs (MALI)	600
# FPGAs	1
FPGA Model 1 Freqs (not in PRRs) (Arria10GX)	100
FPGA: # PRRs	1 2 3 4 5 6
FPGA: Specific Devices (IDs)	1 2 3 4 5 6 7 8 9 (according to Tab. 6)
FPGA: HW Task Implementation	1 2 3 (according to Tab. 7)

points. In the experimentation scenario, we derive Nadir point using the higher individual values for $Time$, $Energy$, and $\frac{1.0}{Utilization}$ from the PF solutions gathered during the respective exhaustive search. The use of $Utilization$'s inverse allows us to treat the experiment as a minimum-goal in all objective functions. Thus, the point $(max(Time), max(Energy), max(\frac{1.0}{Utilization})$ represents the reference point r.

The experimentation scenario 3 aims to demonstrate the use of different architecture types within the DSE environment. Moreover, it augments the design space's size (mainly the mapping possibilities) since more applications raise the number of possible mappings combinations. We run this scenario ten times, detailing the logs resulted from the second run since it produces the highest HV value. The remaining runs we use to analyze the DSE times and their behavior using the HV indicator. We calculate the Nadir point, in this case, using the maximum value in each dimension ($time$, $energy$, and $\frac{1.0}{utilization}$) considering all PF approximations – since we do not execute an exhaustive search corresponding to this scenario.

6.3.3 Results of Case Study #4 – FEHetSS in an Heuristic-based DSE

This section presents the results related to Case Study #4. Regarding **Scenario 1**, Fig. 60 (B to F) shows 3D (with dimensions Time, Energy, and Utilization) scatter plots for five runs of the DSE environment considering the same input parameters. Time measures the last finished application. Energy is for all system PEs. Utilization corresponds to an average between the PEs. Fig. 60 (A) presents the PF of the respective exhaustive search (Sec. 6.2, Case Study #3, Scenario 1) for comparative sake. The plots show the PF approximations' points (marked with "pf" in the legends). It also

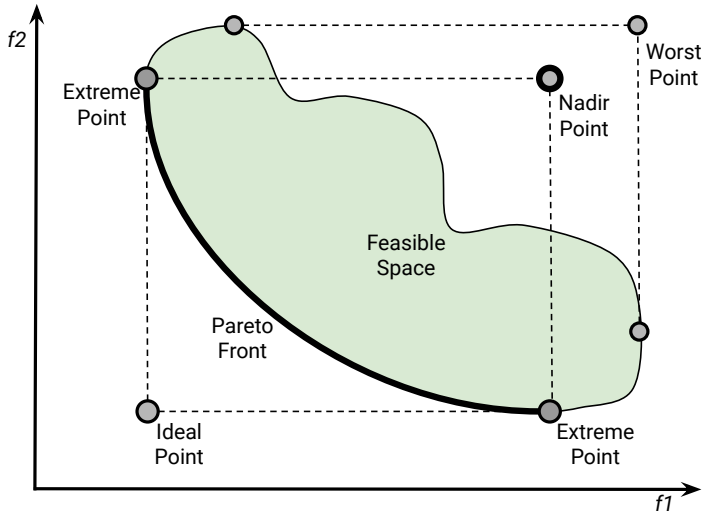


Figure 59 – Representation of a Nadir Point in a Two-Objective Space (based on (WANG; HE; YAO, 2017))

describes the application's mapping and the number of solutions with the same profile in the PF approximation. Besides, Fig. 60 (B to F) presents the "best" solutions that the heuristic has find during its processing. From a certain point of view, these "best" solutions indicate the heuristic's main path through the DSE. Simulated annealing (SA) implementation uses randomness to create solutions and their neighbors. Thus, each DSE produces different solutions and, consequently, PF approximations. Visually, we can observe the higher number of solutions in the approximation of runs 12, 1, and 24 (B, C, and F in Fig. 60). However, we notice that run 24 (F in Fig. 60) provides solutions closest to the ideal point.

Using the DSE environment, we perform 30 runs of Case Study #4, Scenario 1. Fig. 61 presents a bar chart describing the HV (Hypervolume) indicator (presented in Sec. 6.3.2) of the PF and all DSE approximations. Regarding the HR (Hyperarea Ratio) indicator (described in Sec. 6.3.2), Fig. 62 (A) presents a box plot describing the DSE's PF approximations, followed by their statistics (B). Recalling, we use the PF extracted from Case Study #3 – Scenario 1 (Sec. 6.2) – acting as PF in the HR calculation. Besides, we use the inverse of the Utilization metric in the HV and HR calculations. It aims the minimization for all objective functions. The five DSE runs in Fig. 60 (B-F) correspond to approximately the 15th, 25th, 50th, 75th, and 90th percentiles of the HR indicator regarding all the (30) DSE's PF approximations. Therefore, we arrange the plots B to F (Fig. 60) in ascending order (worst to better) according to the DSE's performance indicators (HV and HR), corroborating with the visual insight of the approximations. Box plot of Fig. 62 indicates that 50% of the DSE PF approximations obtain values between 0.94 and 1.16 (1st and 3rd quartiles) for HR indicator. We consider these values as representing good solutions compared to the PF. Regarding the simulation times, Fig.

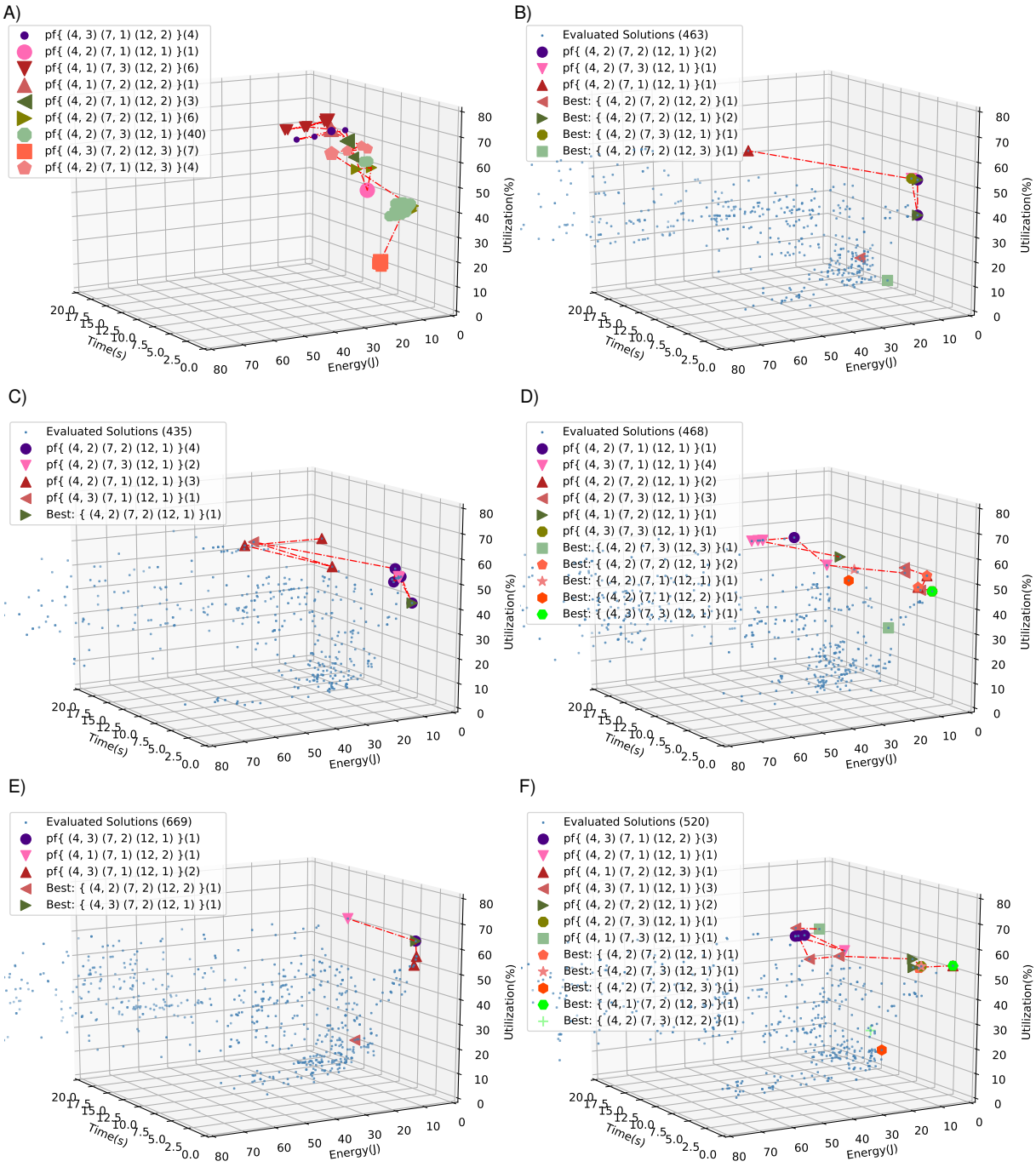


Figure 60 – (A) Pareto Front from Exhaustive Search and (B-F) DSE Runs 14, 12, 1, 27, and 24 (≈ the 10th, 25th, 50th, 75th and 95th percentiles of the HR values regarding the DSE PF Approximations, respectively) considering the Applications 4, 7, and 12.

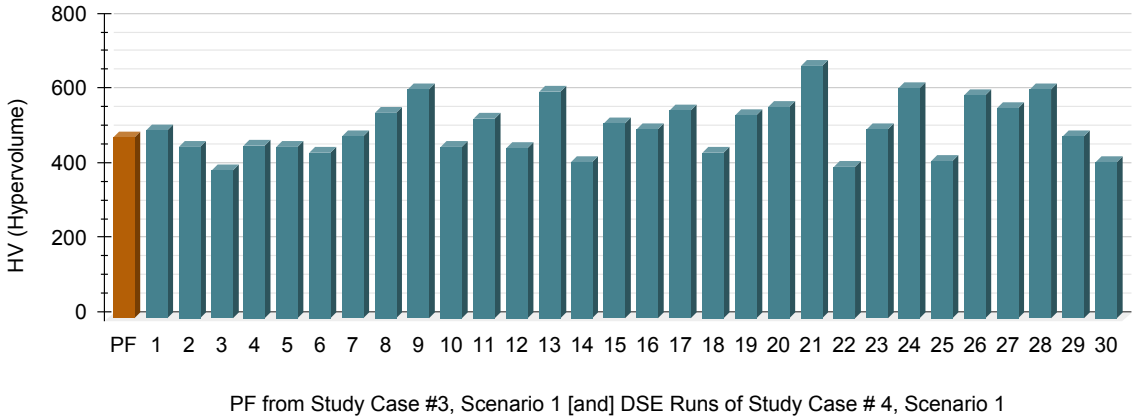
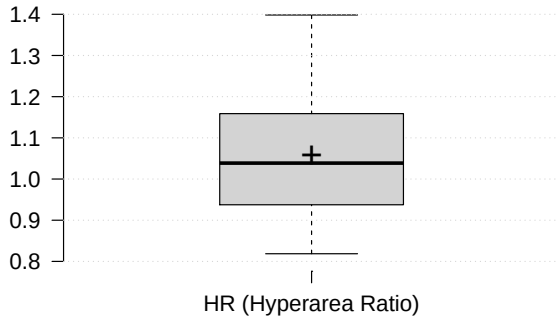


Figure 61 – Bar Plot of the HV (Hypervolume) Indicator for the Exhaustive Search PF and the DSE Runs #1 to #30 (Applications 4, 7, and 12).

A)



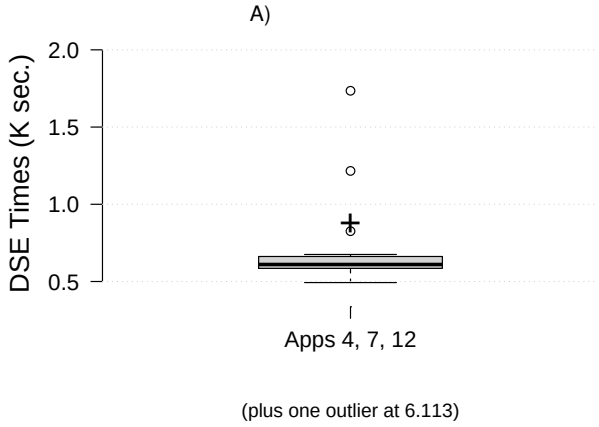
B)

	HR (Hyperarea Ratio)
Upper whisker	1.40
3rd quartile	1.16
Median	1.04
1st quartile	0.94
Lower whisker	0.82
Nr. of data points	30.00
Mean	1.05

Figure 62 – (A) Boxplot of the HR (Hyperarea Ratio) Indicator of the DSE Runs #1 to #30 and using the Case Study #3, Scenario 1 PF as the known Design Space PF. (B) Boxplot Statistics (Applications 4, 7, and 12)

63 presents the respective box plot (A) and their statistics (B).

Regarding the Case Study #4 - **Scenario 2**, Fig. 64 shows scatter plots (B-F) resulting from five DSE runs according to parameters described in Sec. 6.3. We run this DSE scenario 15 times. Fig. 64 also includes the PF plot (A) resulting from the exhaustive search of Case Study #3 - Scenario 2 (described in Sec. 6.2). We extract the HV indicator for the 15 DSE runs (Fig. 65) and the respective HR values (Fig. 66 A). The scatter plots B to F in Fig. 64 shows the runs 14, 15, 13, 7, and 12, corresponding to approximately the 14th, 28th, 50th, 72th, and 92nd percentiles of the HR values regarding the DSE PF approximations. We notice the higher variation in the PF approximations' mapping profiles (B-F) comparing to PF (A). However, the DSE runs provide good PF approximations as indicates Fig. 65. It shows a bar plot of the HV indicators that correspond to the 15 PF approximations generated by the DSE runs and the PF produced by the respective exhaustive search (following the same steps described for Case Study #4 - Scenario 1). On the other hand, Fig. 66 shows a box plot of the HR (Hyperarea Ratio) indicator corresponding to the PF approximations. We notice that all HR values are close to one (1), indicating quality approximations. Fig. 67



	Apps 4, 7, 12
Upper whisker	0.68
3rd quartile	0.66
Median	0.61
1st quartile	0.58
Lower whisker	0.49

Figure 63 – (A) Box Plot of the Simulation Times of Case Study #4 Scenario 1 (Applications 4, 7, and 12). (B) Box Plot Statistics.

presents the simulation times of the DSE scenario 2 – box plot (A) and their statistics (B).

Considering scenarios 1 and 2, we claim the answer to the question "*Can a designer employs FEHetSS as a solutions' evaluator into a Heuristic-based DSE Environment that produces good PF approximations of the design space?*" is *Yes*. Figures 60 and 64 show DSE outcomes highlighting some runs samples. In both scenarios, if we compare with the exhaustive search output (PF), the HV and HR indicators demonstrate DSE PF approximations close to the design space PF. The Simulated Annealing heuristic is simple to deploy, providing a feasible DSE strategy to generate PF approximations. In scenario 1, DSE runs took on average ≈ 0.17 hours to produce an approximation. The respective exhaustive search demands 25.51 hours. Thus, DSE demand ($\approx \frac{0.169}{25.51}$) 0.66223% of the time providing solutions covering at least 82% (lower whisker in Fig. 62) of the PF's design space. Regarding scenario 2, DSE runs on average were ≈ 2.31 hours long. It is $\approx$0.65308% ($\frac{2.31}{353.41}$) of the exhaustive search time. In this case, the approximations cover at least 97% (lower whisker in Fig. 66) of the PF's design space. It is worth mentioning that Simulated Annealing (SA) is only one possible heuristic to employ in DSE between a list of alternatives.

Fig. 68 shows a scatter plots derived from the Case Study #4 - **Scenario 3**, specifically run two (2). According to Sec. 6.3 and Tab. 16, this scenario extends the input parameters allowing the DSE environment to employ any combination of the architecture elements available in the VP repository, but always including an FPGA device. Also, the increased number of applications raises the possible mapping combinations.

Plot (A) of Fig. 68 shows all evaluated solutions and detach the PF and "best" solutions. In a zoomed view, item (B) highlights the profiles of PF and "best" solutions showing the PE types in the architecture (1 for CPU; 2 for GPU; and 3 for FPGA) and the respective models – 1 (A7) and 2 (A15) for CPU, 1 (Mali) for GPU, and 1 (Arria10GX) for FPGA. The "best" solutions detach the DSE choices during its processing,

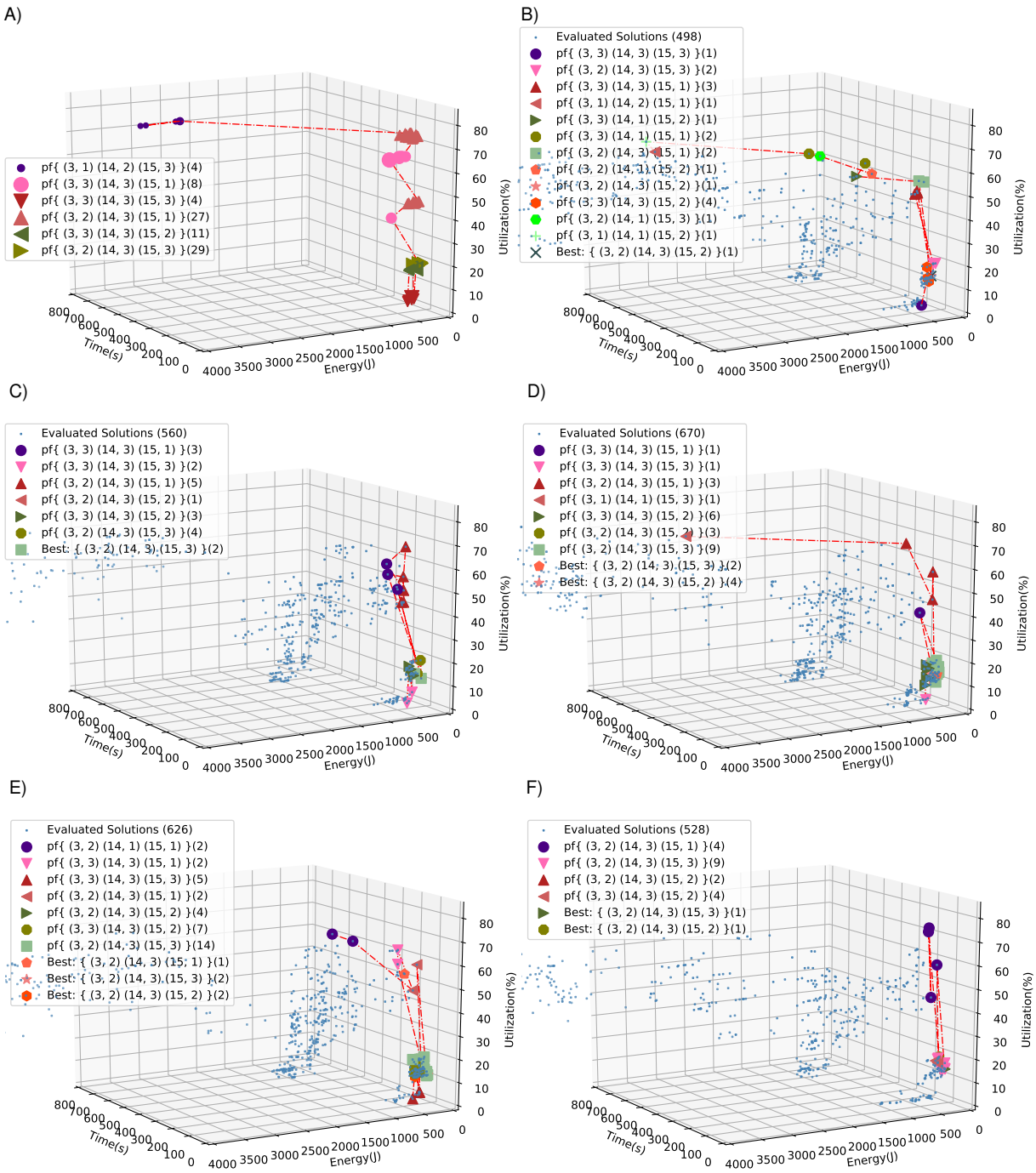


Figure 64 – (A) Exhaustive Search and (B-F) DSE Runs 14, 15, 13, 7, and 12 (≈ the 14th, 28th, 50th, 72th, and 92th percentiles of the HR values regarding the DSE PF Approximations, respectively) for Applications 3, 14, and 15.

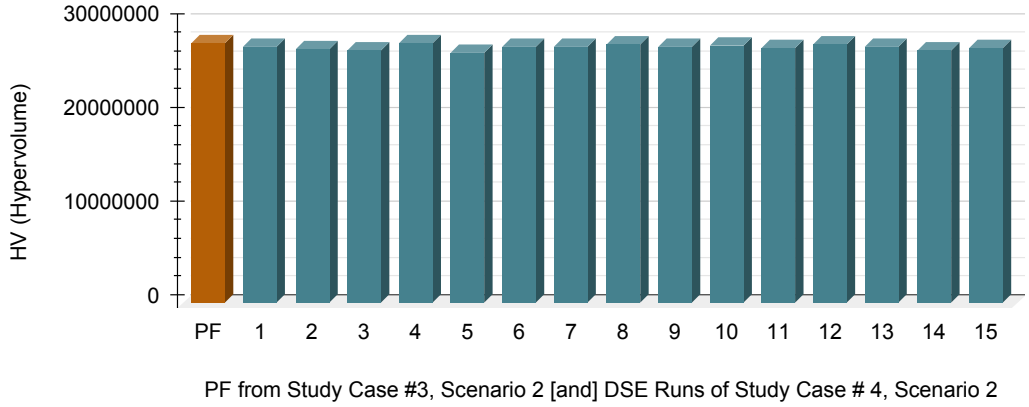
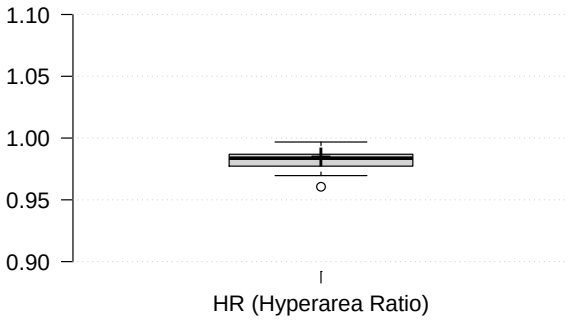


Figure 65 – Bar Plot of the HV (Hypervolume) Indicator for the Exhaustive Search PF and the DSE Runs #1 to #15 (Applications 3, 14, and 15).

A)



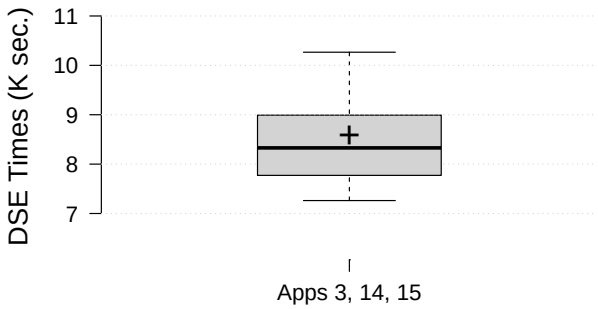
B)

	HR (Hyperarea Ratio)
Upper whisker	1.00
3rd quartile	0.99
Median	0.98
1st quartile	0.98
Lower whisker	0.97
Nr. of data points	15.00
Mean	0.98

Figure 66 – (A) Boxplot of the HR (Hyperarea Ratio) Indicator of the DSE Runs #1 to #15 and using the Case Study #3, Scenario 2 PF as the known Design Space PF. (B) Box plot Statistics (Applications 3, 14, and 15)

being the (24) solutions accepted by the SA heuristic. The SA strategy aims for solutions with lower evaluations (according to Eq. 8). Thus, it makes the SA path trending to the DSE's PF generation. There are 22 solutions of different profiles in the PF interconnected by a red dashed line in Fig. 68 (B). About the Applications' Mapping, plot (C) shows the "Best" solutions. We notice the variability of the solutions generated by the heuristic. It derives from the heuristic experimenting with the possible mappings for each application. Detailing the PF about the FPGA device characteristics, Fig. 68 (D) shows the PF plot. Each profile begins by describing the FPGA device (Tab. 6) and the number of PRRs on it (e.g., 4-5: device 4 with 5 PRRs). Thus, we notice that device 7 (according to Tab. 6) appears in $\frac{8}{22}$ solutions. Analyzing the PF plot, we observe the profiles with devices 9 and 8, beyond employing 3 to 6 PRRs, providing points with good energy consumption and low times. Next, the profile describes each application's mapping (PE: 1 for CPU, 2 for GPU, and 3 for FPGA) and HW Task implementation (only for FPGA) – 1, 2, or 3, according to Tab. 7, e.g., (1, 3, 2): application 1, mapping 3 (FPGA), and HW Task Impl. 2 (2-loop unrolling). Thus, regarding the kernel(s)

A)



B)

	Apps 3, 14, 15
Upper whisker	10.27
3rd quartile	8.99
Median	8.33
1st quartile	7.77
Lower whisker	7.26

Figure 67 – (A) Box Plot of the Simulation Times of Case Study #4 Scenario 1 (Applications 3, 14, and 15). (B) Box Plot Statistics.

Table 17 – Characterization of DSE Runs of Case Study #4, Scenario 3

Run	Evaluated Solutions	Solutions in PF	HV	HV Rank	DSE Time (h)	Evaluated Solutions per Minute
1	4604	17	28.42	3	1.91	40.17
2	6559	22	28.68	1	2.62	41.72
3	4958	19	27.67	7	2.01	41.11
4	4609	20	27.78	6	1.90	40.43
5	5030	23	26.96	8	2.05	40.89
6	6005	28	26.34	9	2.55	39.25
7	5710	25	27.85	5	2.37	40.15
8	5255	25	28.46	2	2.21	39.63
9	5858	36	26.03	10	2.48	39.37
10	3986	17	28.15	4	1.67	39.78
mean	*5257.40*	*22.18*	*27.63*	*n/a*	*2.18*	*40.25*
standard deviation	*776.30*	*5.77*	*0.91*	*n/a*	*0.32*	*0.80*
variance	*602644.93*	*33.29*	*0.83*	*n/a*	*0.10*	*0.64*

mapping, the DSE points out that applications 1 and 8 better match FPGA, while the kernel of applications 2, 5, and 10 also includes the CPU mapping as a good choice. Application 6 can take benefit from all three PE types.

Tab. 17 shows the characterization of scenario 3 with their ten DSE runs. It includes columns for the number of evaluated solutions and those of which are in the PF approximation. Also, it presents the HV values and their rank (better to worst), and the DSE times. Besides, Fig. 69 presents box plots describing the runs in terms of the number of evaluated solutions and those in the PF, as the HV indicator and DSE times.

We regard the better value for the HV indicator for run 2. Considering a common Nadir point, run 2 achieves a PF approximation that ”dominates” a larger volume of the design space comparing the other runs (Fig. 70). However, the difference is not so significant. Moreover, according to Fig. 71, we identify no relation between the number of evaluated solutions and those in the PF. The HV values present little variation among the runs, although the number of PF’s solutions does a greater diversification. The SA randomness and their exploring strategy produce diversified solution sets. However, we consider the DSE runs’ performances very similar.

The answer for the question(s) ”*Can FEHetSS be integrated into a Heuristic-based DSE Environment providing diversified quality solutions? Also, demonstrating the ex-*

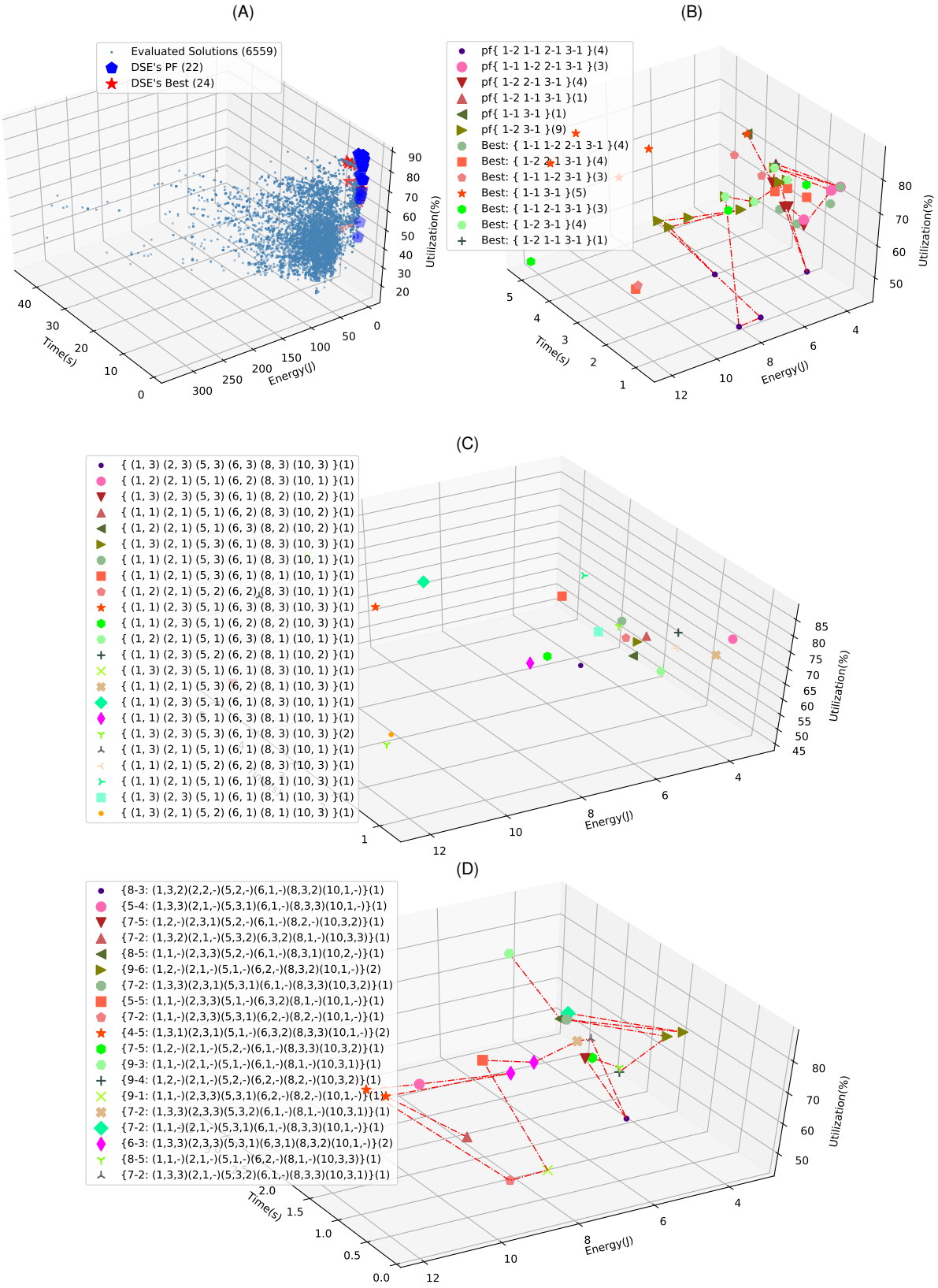


Figure 68 – DSE for Applications 1, 2, 5, 6, 8, and 10: (A) All Evaluated Solutions Highlighting the PF and DSE's "Best" Solutions. (B) Architecture Elements of PF and DSE's "Best" Solutions. (C) "Best" Solutions Detaching the Applications Mappings. (D) PF Solutions Indicating the Specific FPGA Device and Number of PRRs, followed by the Appl. ID (Tab. 5, its Kernel Mapping (1: CPU, 2: GPU, and 3: FPGA), and the used HW Task Implementation for FPGA (Tab. 7).

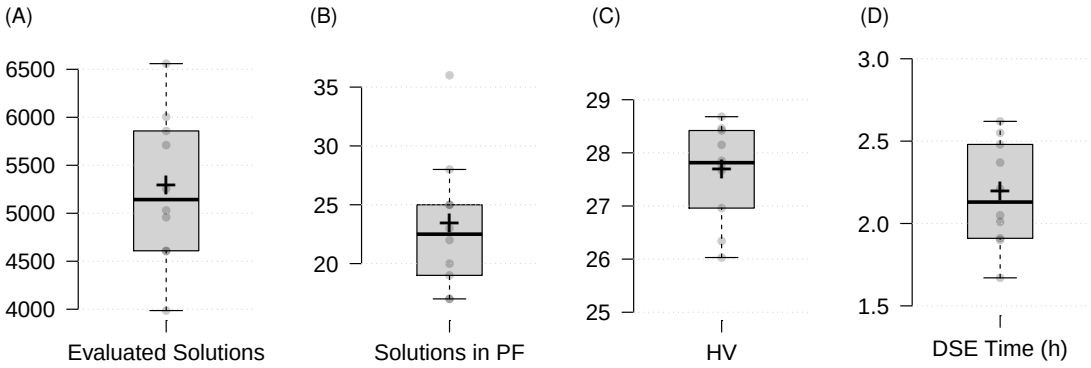


Figure 69 – Box Plots for (A) Evaluated Solutions, (B) Solutions in PF, (C) HV Indicator, and (D) DSE Time for Case Study #4, Scenario 3.

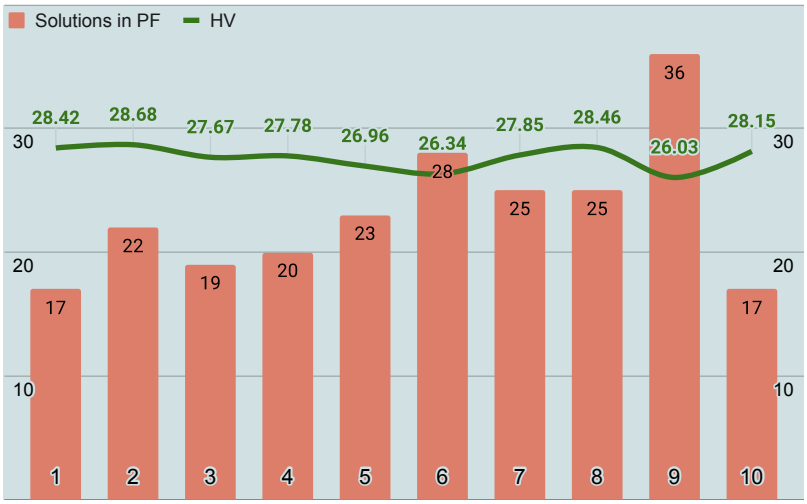


Figure 70 – No Relation Between the PF's Size and the HV Indicator (Case Study #4, Scenario 3).

ploration of the architecture features during the solutions quest?" is *Yes*. Our claim is based on the following observations:

- The solutions' diversity of the PF and "best" sets provided by the SA heuristic. Both the solutions' sets on the PF and those called "best" present different types of architecture (according to Fig. 68 – B), employing varied combinations of PEs;
- In addition, DSE uses varied mappings for application kernels during its execution (according to Fig. 68 – C);
- Furthermore, in cases of mapping to FPGA, the implementations available as HW tasks provide a more in-depth assessment of the use of HW accelerators, including the experimentation with devices equipped with different amounts of HW resources organized in a number of PRRs (Fig. 68 – D);
- Figures 69 (C) and 70 shows that DSE produce similar PF approximations, independent of the sets' cardinality;

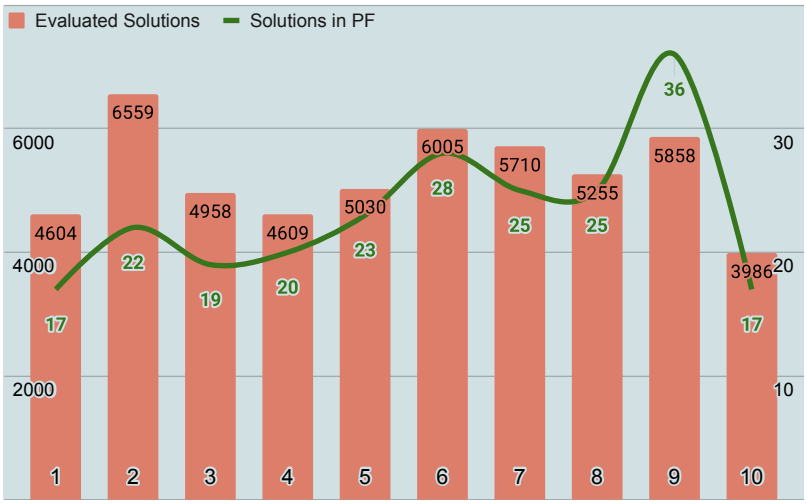


Figure 71 – No Relation Between the Sets Cardinalities of Evaluated Solutions and Solutions in PF (Case Study #4, Scenario 3).

- Tab. 17 shows that FEHetSS simulates several solutions per minute (≈ 40), providing rapid evaluation results;

6.4 Chapter Summary

This chapter has presented the Case Studies #3 and #4. We organize the presentation by Case Study using sections 6.2 and 6.3 for that sake. In Case Study#3, we performed exhaustive searches in restricted design spaces. During Case Study #4, those same design spaces were explored by a heuristic-based DSE environment integrating FEHetSS for design points evaluation. We have employed Simulated Annealing as a heuristic strategy to traverse the design space. At the end of each section presenting the results (Sections 6.2.1 and 6.3.3), we answered the respective questions described in Tab. 14 to the achievement of each case study's goal.

7 ASSESSING THE RESEARCH CHALLENGES

This chapter evaluates the modeling methodology and the FEHetSS simulator regarding the Research Challenges presented in Sec. 3.3. We follow the modeling steps described in Sec. 4.3 creating models (or portion of) that feed a VP repository. Attached to the repository, we prepare a set of automation scripts. Using the repository and scripts, we can define a VP according to each case study through a string representation argument. The string follows the VP description structure described in Fig. 55. This structure enables/facilitates the creation of solutions in the form of XML files. FEHetSS simulator receives an argument indicating the (XML) file containing the virtual platform (VP) to simulate. Thus, all these facilities support an easy interaction during the creation and evaluation of the case study's design points.

Previously, we evaluate the results of each case study answering the respective questions defined in Tables 4 (in Sec. 5.1) and Tab. 14 (Sec. 6.1). Tab. 18 briefs those questions and theirs answers. Based on these answers and the presented results in sections 5.2.1, 5.3.1, 6.2.1, and 6.3.3, we can proceed to evaluate the Thesis' Research Challenges.

7.1 Evaluating Challenge 01: *How to enable early-DSE on heterogeneous systems that include reconfigurable hardware acceleration through FPGAs?*

The proposed methodology (illustrated by Fig. 34 in Sec. 4.3) describes how to make estimations/definitions about virtual platforms (VPs). Specifically for VPs including FPGA devices, we use an OpenCL HLS tool (INTEL, 2020a), through its `aoc` compiler, providing several estimates suitable for annotating our simulator. These annotations include data for latency and a maximum frequency of the application's kernel(s) implemented as an HW task(s). Besides, data about the HW resources necessary for it. Moreover, `aoc` generates HW design files (such as quartus project file - ".qpf") that a designer can configure aiming to subsequently estimate power consumption for the device's portions – using PowerPlay tool from Quartus (INTEL, 2020b). The power

Table 18 – Briefing of the Case Studies' Questions

Case Study	Scenario(s)	Question	Answer
#1	1	Based on high-level models, can FEHetSS provide appropriate log metrics in heterogeneous architectures' evaluation during a (manual) DSE?	Yes. FEHetSS simulates the settings and produces metrics to appropriately evaluate the design points. The case study highlights the benefits of using HW accelerators, especially if employing optimized implementations for an HW task.
#2	1	Can FEHetSS be used to assess different HW task implementations based on high-level models describing the tasks themselves and the HW accelerator device (FPGA)?	Yes. The case study shows that when employing optimized HW task implementations, the FEHetSS models suffer alterations on its annotations, changing the HW resources, power consumption, and latency of a kernel. Thus, simple modifications in the VP models allow producing valuable system design outcomes.
#2	2	Can FEHetSS be used to assess different HW task implementations based on high-level models describing the tasks themselves and the HW accelerator devices (FPGA) with diversified resources?	Yes. In this scenario, we employ a series of nine FPGA models combining them with the seven implementations of a kernel (2MM). We observe that some combinations are not feasible. Also, the experiment allows the assessment of an important aspect: What device "size" and HW task implementation better match each other?
#3	1, 2	Can FEHetSS be used to perform Exhaustive Searches in a feasible time, providing suitable Metrics to identify the design space's PF?	Yes, but it depends on the design space size, simulation demand, and outcome breadth. If a designer can make their evaluation based on a "reduced" design space, the time to pay in simulation and searching may be worth it. Thus, there is a trade-off between simulation time and optimal PF identification.
#4	1, 2	Can FEHetSS be employed as a solutions' evaluator into a Heuristic-based DSE Environment that produces good PF approximations of the design space?	Yes. In this case study, both DSE scenarios produce PF sets comparable to those generated in the respective exhaustive searches but taking less than one percent of the time. Simulated Annealing is simple to deploy, providing a feasible strategy to generate PF approximations in a DSE environment.
#4	3	Can FEHetSS be integrated into a Heuristic-based DSE Environment providing diversified quality solutions? Also, demonstrating the exploration of the architecture features during the solutions quest?	Yes. This case study shows a diversified set of outcomes generated by the DSE. We also notice a spread variation in the architecture components and the application mappings. Other DSE parameters, such as the number of PRRs and specific devices, have demonstrated the broad scattering during the heuristic strategy. The DSE runs produced similar PF approximations, showing uniformity between the simulations. We evaluate the quality of the PF approximations using the HV indicator based on a common reference point. The results corroborate with the quality solutions idea.

annotations describe the PRR portions employed for the HW task implementations. Also describing the board interface and periphery HW, both latter playing a static role (configured once) in the FPGA device.

Considering these annotations, FEHetSS employs a power model (described in Alg. 2) that computes the power regarding each PRR (in any of their states), the board interface, and the periphery hardware. Moreover, using an `aoc` compiler report, we define a set of equations (eq. 2 to 6) aiming to estimate the kernel(s) latency implemented as an HW task.

Case studies #1 (Sec. 5.2) and #4 (Sec. 5.3) deal directly with this challenge by employing FEHetSS and solutions created through the VP Repository. Case study #1 makes the exploration manually, while #4 uses a heuristic-based DSE environment. The results of both cases corroborate the accomplishment of challenge 01 by showing the solutions in a three-dimensional design space, highlighting the Pareto front (PF). It

provides designers a panoramic view enabling them to decide the design directions.

7.2 Evaluating Challenge 02: *How to assess (at early stages) different architectural FPGA implementations for a specific application kernel, as an HW task, within a heterogeneous system?*

Considering a specific HW design of an application kernel, the second case study (Sec. 5.3) presents the attending of the challenge 02. The modeling steps allow creating VPs characterizing optimizations such as multiple compute units and loop unrolling. The VP simulations in FEHetSS describe the impact of each optimization on the produced metrics. Multiple compute units increase the used HW resources and the power consumption. In turn, it decreases the number of work-items to be processed by each HW module. Further, loop unrolling also increases power consumption and HW resources but diminishes the kernel(s) loop iterations. Thus, the optimizations affect all logged metrics, decreasing the execution time and increasing power and utilization, featuring a multi-objective optimization problem.

In addition to the optimized HW tasks, scenario 2 of Case Study #2 provides the opportunity to evaluate an HW/SW interaction by combining devices with different amounts of available logical blocks. Although each HW implementation does not change from one device to another, each device's capacity entails its pressure by power, reflecting on the general device consumption. Moreover, since the HW demand of the tasks varies according to the optimization parameters, scenario 2 shows that the modeling assumptions and FEHetSS simulations allow evaluating the appropriate device's "size" given an experimentation case. The presented analysis for performance and power are richer than only evaluate the HW task of the kernel in an isolated manner. Modeling the platform equipped with a CPU and an FPGA, also attending a specific load of an application (main tasks and elaboration task), we can consider the actual demand of the kernel against the system reality. An evaluation of only the kernel's HW does not carry this information. Thus, with few modifications in the VPs models, FEHetSS provides metrics regarding the optimized HW tasks and the employed devices, allowing to evaluate the trade-off between them.

7.3 Evaluating Challenge 03: *How to model FPGA units at a high level, supporting PRR and DPR, focusing on high-level system simulation?*

The methodological steps described in Sec. 4.3 include aspects related to PRR and DPR. The power model (represented by Alg. 2 in Sec. 4.3.1) regards the device par-

tition in PRR by evaluating each one separately, considering its size and task demand for power. In an application model, the task graph embraces a characterization for each task, describing its type and performance model. However, for elaboration tasks mapped to an FPGA, the description includes data about the required HW resources and the core logic (HW task implemented in the FPGA fabric) operation parameters (e.g., powers, frequency, and bitstream size). During a simulation, FEHetSS uses this data to verify a PRR capacity to accommodate the HW task, simulate the reconfiguration process (DPR) when necessary and the execution itself, considering latency and power consumption. The FPGA modeling regards the number of PRR to employ in the device, besides the board interface and the periphery HW such as transceivers and I/O components. Thus, the modeling abstracts the device as being a component aggregation. Consequently, during the simulation, each component influence the virtual platform, contributing to the system's metrics.

7.4 Chapter's Considerations

In this chapter, we have presented an evaluation of the Research Challenges of the Thesis based on the answered questions and results of the performed Case Studies. Tab. 18 shows a brief version of each pair question/answer. It helps us to evaluate the Research Challenges.

In general, there are some features that we can highlight about FEHetSS and the modeling methodology.

- FEHetSS provides rapid simulations producing metrics suitable to evaluate the VP performance. The case study #3 shows such capacity by performing exhaustive searching in restricted design spaces but evaluating more than one hundred thousand solutions.

- Some solutions embrace *VirtualHWAccel*s as FPGA devices, including multiple PRR and considering the DPR process. FEHetSS produces metrics that account for the PRR partitions and considers the required time/power to perform the device/PRR configuration.

- The following FEHetSS's metrics present some interesting characteristics:

  - *time*, shows the beginning/ending of each task (main or elaboration) of the running applications.

  - *power*, describes the power consumption of each PE, including CPU, GPU, and FPGA types.

  - *energy*, based on the previous two metrics, we can calculate the consumed energy of each PE and for the whole system.

  - *utilization*, shows how much used is a PE during the simulation. In the case of an FPGA, utilization measures the allocated HW resources compared to those available in the FPGA device (taking the number of ALUTs as the measuring unit).

- The modeling methodology allows considering optimization aspects for the HW task implementations. Through OpenCL pragmas and tooling, we annotate models complying with the aimed optimizations producing outputs subsiding designs comparison.

8 CONCLUSIONS

This work has presented a Modeling and Simulation Infrastructure for FPGA-enabled Heterogeneous Systems. The Modeling methodology includes the steps to create high-level models describing the systems' architecture and workload. We call these models Virtual Platforms (VPs), employing XML files to specify them. Architecture models incorporate each PE describing the operating frequency, power, and idle power. For FPGA, the model separates the power model in core fabric and periphery HW, also including the device's HW resources, board interface resources, and operational parameters. For the workload, a task graph depicts the applications. Each node contains a performance model indicating the frequency and latency. For an HW task implemented in the FPGA fabric, nodes include its HW resources, power model for the core logic, and its bitstream size. The edges define the simulation's sequence. It offers forward, backward, and branch transitions – the latter allowing advance for an elaboration task which allows exploring the system heterogeneity by being executed in any of the PEs available in the architecture. We presented the flow and supporting tools that meet the VPs model's creation. Besides, we describe a power model for the *VirtualHWAccel*s considering estimations from such SW (e.g., Quartus). Moreover, we employed the reports from an HLS tool (`aoc` compiler) to model the HW components and the latency aspects. Thus, this modeling framework provides appropriate elements and properties to describe a heterogeneous system even in the presence of a PRR-partitioned Dynamically Reconfigurable *VirtualHWAccel* described as an FPGA unit.

In compliance with the modeling methodology, we presented a System-Level Simulator named FEHetSS. It allows VP assessment in early Design Space Exploration (DSE). We described the FEHetSS's structure highlighting the FPGA processing element (PE) feature. As soon as a VP arrives, FEHetSS creates the PEs and the applications. For each PE, FEHetSS generates a module representing its behavior. A workload generator module uses the applications' models to push their tasks' threads to simulate in a PE through the system-calls emulator, passing by a communication channel till the target processor which, in the case of elaboration tasks, can be a CPU,

a GPU, or an FPGA.

Regarding the *VirtualHWAccel*s, an FPGA model includes details about their partition in PRRs and DPR processing. FEHetSS simulates the reconfiguration time and the respective power, passing the PRR just configured to a different power profile. The simulation flow obeys mode-based device transitions as well as PRR defined in the FPGA fabric.

We consider both FEHetSS and Modeling Methodology dealing with the essential elements for a VP simulation providing suitable model elements and properties. FEHetSS also produces valuable metrics supporting designers in decision-making in early design duties. We performed four case studies to detach both the modeling methodology and FEHetSS simulator capabilities:

- (i) design point evaluation tool during an early DSE;
- (ii) HW designs evaluation tool;
- (iii) rapid simulation framework during exhaustive searches; and
- (iv) setting metrics generation tool embedded in a Heuristic-based DSE environment.

Using FEHetSS as a **DSE evaluation tool** allows a designer to assess different architecture setups – e.g., homogeneous or heterogeneous, GPU or FPGA accelerated, employing or not employing DPR and multiple PRRs – and considering specific workloads with different mappings for the applications' kernels. Conforming Sec. 5.2.1, FEHetSS is capable of generating evaluation metrics that can be analyzed by searching for the *Pareto*'s front of the design space.

FEHetSS provides high-level evaluation of **HW design** alternatives. We annotated the simulator models with estimations provided by the supporting tools applying OpenCL pragmas to optimize applications' kernels. Further, we used the simulation and its results to assess the optimizations' impact through FEHetSS's metrics – such as execution time, power consumption, and PE (HW) utilization. Besides, as presented in Sec. 5.3.1, FEHetSS was also charged with models of a variety of devices from the Arria10GX FPGA family. It allowed a trade-off evaluation of what better amount of logical resources necessary in an FPGA unit while accomplishing HW design implementations of an application kernel (2MM).

The rapid FEHetSS simulations offer opportunities to perform **exhaustive searches** in a design space. The required time for a exhaustive space exploration is not always a feasible choice. However, in the case of restricted design spaces, the trade-off may be worth it. The design space restriction can be the use of a lower range of frequencies, a reduced set of devices, a lower number of HW task implementations,

or even a combination of factors. In Sec. 6.2.1, two exhaustive searches produced design space explorations considering a reduced range of the search parameters.

Regarding a less computationally demanding workload, the first scenario finishes the examination after ≈ 25 hours while the second scenario took circa 15 days in the exploring. Aiming to shrink the simulation time, a designer could model an application containing nearly only its kernel, simulating it in FEHetSS in a full exploration context. It could allow a relevant evaluation, possibly in a reduced simulation time.

As an independent simulator, we plugged FEHetSS in a heuristic-based DSE environment performing the **generation of settings' evaluation metrics**. It is a part of a DSE process following the Y-Chart approach. The environment's solutions generator demanded our VP repository feeding settings to the environment's simulations through several FEHetSS's threads. The output metrics passed to the DSE solution evaluator generate evaluation values for each setting. The environment's heuristic used them on their strategy to accept (or not) new solutions. Sec. 6.3.3 has presented the results of using FEHetSS, the VP repository, and the simulated annealing as an optimization heuristic. Compared to the exhaustive search Pareto front, the first two scenarios using the DSE environment cover at least 82% and 97% of the respective design spaces, using $\approx 0.66\%$ and 0.65% of the full search time, respectively. Scenario 3 showed the DSE environment exploring a vast design space. Summing all possible configurations in this scenario, we obtained 594712368 different settings. The average number of evaluated solutions by the DSE runs was ≈ 5257 (0.0009% of the possible configurations). Between the ten DSE runs in scenario 3, the second presented the best value for the HV indicator – considering a common Nadir point for all approximations. However, all runs showed nearby values for HV, indicating similar performances of the DSE environment.

8.1 Answering the Research Challenges: Thesis' Contributions

We delineated three research challenges for the Thesis. These challenges directed the work through a series of steps involving aspects such as:

- Adoption of modeling elements based on the OpenCL concepts – e.g., host and device processors, applications' structure model. Besides, the use of OpenCL codes from a benchmark (Polybench) serving as workload applications;
- Identification of HW tools capable to provide estimations for FPGA latency&power profiles (`aoc` and *Quartus*);
- Utilization of real HW measures aiming to profile the power of CPU and GPU processors, beyond the tasks' latencies;
- Definition of a methodology to describe Virtual Platforms containing Architecture and Workload models. The methodology included steps and elements related to:

  - FPGA device power estimations, e.g., static (board interface, periphery HW) and dynamic (PRRs) portions;
  - HW task modeling;
  - PRR partitioning;
  - DPR reconfiguration process;
  - Kernel/blocks latencies;

- Implementation of the model conceptions through FEHetSS System-Level Simulator;

- Preparation of a Virtual Platform (VP) models repository and a VP creation interface (automation scripts);

Thus, considering the developed work, we answer the **research challenges** in the following paragraphs presenting the *main contributions* of the Thesis.

(Challenge 01) **How to enable early-DSE on heterogeneous systems that include reconfigurable hardware acceleration through FPGAs**?

We defend that our methodology provides the flow steps and model elements describing models of heterogeneous systems. It also includes features of reconfigurable HW described as FPGA. High-level models describing an FPGA unit embraces reconfiguration properties by considering PRR partition and DPR processing. Task-level descriptions used in the task-graphs allow modeling an application considering its setup/management portions (main tasks) and core parts (elaboration tasks). Thus, the Modeling&Simulation Infrastructure provides the essential components to fulfill the challenge. FEHetSS implements those components as a System-Level Simulator. The presented Case Studies demonstrate the early-DSE capability of FEHetSS by performing exploration in manual (Sec. 5.2)), exhaustive (Sec. 6.2), and heuristic-based (Sec. 6.3) manners.

(Challenge 02) **How to assess (at early stages) different architectural FPGA implementations for a specific application kernel, as an HW task, within a heterogeneous system**?

Our Modeling&Simulation infrastructure includes elements and properties capable of describing architectural details of HW tasks' implementations. Regarding HW task modeling, the annotations for latency and power come from HW tools. It considers the distinct features of the possible realizations. Moreover, an HW task implementation can demand different resources in an FPGA unit. Since a better match between an HW module and its chosen device also concerns a system designer, our device model properties permit us to describe each specific device, characterizing it in terms of available resources and power profile. Our second case study (Sec. 5.3) presented an evaluation showing that optimizations like loop unrolling and multiple compute units

affect the kernels' implementations, impacting the overall performance. Additionally, the second part of the case study evaluates the system-level effects regarding the combinations of seven (7) different HW task implementations with nine (9) samples of an FPGA device family (Arria10GX). Therefore, we advocate that our model structures and properties attend the challenge.

(Challenge 03) **How to model FPGA units at a high level, supporting PRR and DPR, focusing on high-level system simulation**?

According to the presented Modeling&Simulation infrastructure, we modeled FPGA as an HW device containing a PRRs set beyond other static components. Each PRR has its available HW resources property. On the other side, an HW task requires some HW resources amount, being modeled by appropriate attributes. Powers, frequency, and partial bitstream are also properties of an HW module. For the simulation of an HW task execution in a not yet configured device's PRR, a (re)configuration must occur first. Thus, during a partial reconfiguration (PR) process, the FEHetSS simulator regards the DPR feature by considering the size of a partial bitstream and operational parameters of the PR controller, simulating its latency and power consumption. Moreover, the power estimation module uses the power profile of the previously configured task and the one that will take its "place" to estimate the power consumption in three time periods: right before the PR, during the PR, and soon after the PR. All performed case studies considered PRR and DPR features, but especially case study 4 – scenario 3 (Sec. 6.3.3) experiments a larger number of possibilities regarding the number of PRRs and applications, causing a lot of reconfigurations during the scenario simulation. We claim that the developed infrastructure deals with PRR and DPR feature attending the related challenge.

8.2 Future Perspectives for the Thesis

During the development of this Thesis, we do not approach some aspects in an ideal or wished way. Thus, we can glimpse some future viewpoints about the enhancement of the Thesis. We use this space to present such panoramas.

We plan to expand the integration of FEHetSS in DSE environments by experiment with other heuristics to direct the exploration. Some alternatives as DSE heuristics are Genetic Algorithms (AN; GAMATIÉ; RUTTEN, 2015; NOGUEIRA et al., 2016), Ant Colony Optimization (DORIGO; STÜTZLE, 2010; WANG et al., 2007), and Particle Swarm Optimization (SINAEI; FATEMI, 2018).

A current weakness of FEHetSS is the lack of an explicit memory contention model, mainly the data transfer between host and device. Currently, FEHetSS considers the communication along with the execution itself. In the tasks' modeling, we include in their latencies memory transfer operations. We intend to incorporate memory aspects

in the virtual platform by modeling and simulating the memory structure. The inclusion of memory details in the VP models would affect the FEHetSS simulator's communication channel module. A memory model can enhance the FEHetSS simulation by giving a more precise system model in terms of overall latency as well as energy consumption by adding a new type of architectural component – the main memory.

In the task-level modeling of the applications, we consider the kernels as a whole, even that the original OpenCL kernel(s) file has more than one kernel function. Thus, in this Thesis, we employ the coarser version of the kernel as a sole HW task in the modeling of the applications' task graphs. As presented in Sec. 5.2.1, we can modify this granularity by defining each kernel's function as a separate elaboration task in the application model. This can impact the size of the PRRs since smaller kernels will require a lower amount of the device's resources, allowing to use of a larger number of PRRs and, consequently, more HW tasks being executed concomitantly. Clearly, this scenario will demand more reconfiguration points during the simulation, even that each reconfiguration will occur more rapidly also demanding less energy. Thus, there is a trade-off between the raising of PRRs and parallel HW tasks, against the higher number of less demanding reconfigurations. We plan to elaborate experiments evaluating this trade-off.

Currently, the CPU model of FEHetSS considers its power in a simplified way. We verify that each application requires its specific energy demands. However, FEHetSS models define only one power profile for a CPU core. It causes the definition of a weighted power estimation (Eq. 7) when simulating multiple applications together. Moreover, a CPU has different demands when executing host tasks and device tasks. Thus, we envision that an application model can bring power annotations detailing each task type apart. Thereby, when running in a CPU processor, an application's *main task* has its power demand, which is different from an *elaboration* one charging the same core type.

Another related aspect that worths research initiative is to enhance FEHetSS aiming to deal with the system's adaptability by allowing it to adapt its PEs mapping strategy respecting the system's goals and restrictions – e.g., maximal performance, energy-saving, and area constraints. Considering an execution scenario, the tasks of workload applications can be started at different moments of a simulation, also employing distinct mappings according to the current system conditions and priorities. The FEHetSS's Orchestrator component can be extended to implement mapping strategies aiming at the architectural resources allocation.

Currently, the preparation of the models in the VP repository occurs manually. It is an onerous task that demands the tooling and modeling skills of a designer, requiring the applying of HLS and HW design tools, possibly with high compilation times. A possible enhancement to FEHetSS is to adapt in the methodology a (rapid) estimation

tool related to performance, power, and area aspects of an FPGA-based PE, as well as CPU and GPU. For FPGA, HAPE (MAKNI et al., 2018) and COMBA (ZHAO et al., 2020) are candidate tools. Regarding the task-level modeling of the FEHetSS applications, by using a task graph model, we envision a tool that automatically recognizes these elements in C/C++/OpenCL codes constructing the correspondent model to employ in the FEHetSS's inputs, also allowing to annotate them with suitable estimations.

REFERENCES

AMAZON. **Amazon EC2 F1 Instances For Educators**. [S.l.]: Amazon Web Services, Inc. or its affiliates, 2020. https://aws.amazon.com/pt/education/F1-instances-for-educators/.

AN, X.; GAMATIÉ, A.; RUTTEN, E. High-level design space exploration for adaptive applications on multiprocessor systems-on-chip. **Journal of Systems Architecture**, [S.l.], v.61, n.3–4, p.172–184, 2015.

ASCIA, G. et al. Performance evaluation of efficient multi-objective evolutionary algorithms for design space exploration of embedded computer systems. **Applied Soft Computing**, [S.l.], v.11, n.1, p.382–398, 2011.

AUDET, C. et al. Performance indicators in multiobjective optimization. **European Journal of Operational Research**, [S.l.], 2020.

BASILI, V. R.; CALDIERA, G.; ROMBACH, H. D. The goal question metric approach. **Encyclopedia of software engineering**, [S.l.], p.528–532, 1994.

BERTELS, K. (Ed.). **Hardware/Software Co-design for Heterogeneous Multi-core Platforms**: The hArtes Toolchain. [S.l.]: Springer Netherlands, 2012. p.1–8.

BETEMPS, C. M. et al. Exploring Heterogeneous Task-Level Parallelism in a BMA Video Coding Application using System-Level Simulation. In: VIII BRAZILIAN SYMPOSIUM ON COMPUTING SYSTEMS ENGINEERING (SBESC), 2018., 2018. **Anais...** IEEE, 2018. p.75–82.

BLOUIN, D. et al. AADL Extension to Model Classical FPGA and FPGA Embedded within a SoC. **International Journal of Reconfigurable Computing**, [S.l.], v.2011, p.15, 2011. Article ID 425401.

BOLCHINI, C. et al. An orchestrated approach to efficiently manage resources in heterogeneous system architectures. In: IEEE INTERNATIONAL CONFERENCE ON COMPUTER DESIGN (ICCD), 2015., 2015. **Anais...** [S.l.: s.n.], 2015. p.200–207.

BONAMY, R.; BILAVARN, S.; CHILLET, D.; SENTIEYS, O. Power consumption models for the use of dynamic and partial reconfiguration. **Microprocessors and Microsystems**, [S.l.], v.38, n.8, Part B, p.860 – 872, 2014.

BRITO, A. V. de; KUEHNLE, M.; MELCHER, E. U. K.; BECKER, J. A General Purpose Partially Reconfigurable Processor Simulator (PReProS). In: IEEE INTERNATIONAL PARALLEL AND DISTRIBUTED PROCESSING SYMPOSIUM, 2007., 2007a. **Anais...** [S.l.: s.n.], 2007a. p.1–7.

BRITO, A. V. de; MELCHER, E. U. K.; ROSAS, W. An open-source tool for simulation of partially reconfigurable systems using SystemC. In: IEEE COMPUTER SOCIETY ANNUAL SYMPOSIUM ON EMERGING VLSI TECHNOLOGIES AND ARCHITECTURES (ISVLSI'06), 2006. **Anais...** [S.l.: s.n.], 2006. p.2 pp.–.

BRITO, A. V. et al. Modelling and Simulation of Dynamic and Partially Reconfigurable Systems using SystemC. In: IEEE COMPUTER SOCIETY ANNUAL SYMPOSIUM ON VLSI (ISVLSI '07), 2007b. **Anais...** [S.l.: s.n.], 2007b. p.35–40.

CAI, L.; GAJSKI, D. Transaction Level Modeling: An Overview. In: IEEE/ACM/IFIP INTERNATIONAL CONFERENCE ON HARDWARE/SOFTWARE CODESIGN AND SYSTEM SYNTHESIS, 1., 2003, New York, NY, USA. **Proceedings...** ACM, 2003. p.19–24. (CODES+ISSS '03).

CALLOU, G. et al. Energy consumption and execution time estimation of embedded system applications. **Microprocessors and Microsystems**, [S.l.], v.35, n.4, p.426–440, 2011.

CAMPOSANO, R.; WILBERG, J. Embedded system design. **Design Automation for Embedded Systems**, [S.l.], v.1, n.1, p.5–50, 1996.

COSTA, D. et al. Characterizing the Polybench/GPU on an MPSoC. In: SOUTH SYMPOSIUM ON MICROELECTRONICS (SIM 2019), 2019., 2019. **Anais...** [S.l.: s.n.], 2019. p.91–94.

DE MICHELI, G.; GUPTA, R. K. Hardware/software co-design. **Proceedings of the IEEE**, [S.l.], v.85, n.3, p.349–365, 1997.

DORIGO, M.; STÜTZLE, T. Ant Colony Optimization: Overview and Recent Advances. In: GENDREAU, M.; POTVIN, J.-Y. (Ed.). **Handbook of Metaheuristics**. Boston, MA: Springer US, 2010. p.227–263.

DUHEM, F.; MULLER, F.; BONAMY, R.; BILAVARN, S. FoRTReSS: a flow for design space exploration of partially reconfigurable systems. **Design Automation for Embedded Systems**, [S.l.], v.19, n.3, p.301–326, Sep 2015.

DURELLI, G. C. et al. Runtime resource management in heterogeneous system architectures: The save approach. In: PARALLEL AND DISTRIBUTED PROCESSING WITH APPLICATIONS (ISPA), 2014 IEEE INTERNATIONAL SYMPOSIUM ON, 2014. **Anais...** [S.l.: s.n.], 2014. p.142–149.

ERBAS, C.; PIMENTEL, A. D.; THOMPSON, M.; POLSTRA, S. A framework for system-level modeling and simulation of embedded systems architectures. **EURASIP Journal on Embedded Syst.**, [S.l.], n.1, p.82123, 2007.

FENG, L.; LIANG, H.; SINHA, S.; ZHANG, W. HeteroSim: A Heterogeneous CPU-FPGA Simulator. **IEEE Computer Architecture Letters**, [S.l.], v.16, n.1, p.38–41, 2017.

FONSECA, C. M.; PAQUETE, L.; LOPEZ-IBANEZ, M. An Improved Dimension-Sweep Algorithm for the Hypervolume Indicator. In: IEEE INTERNATIONAL CONFERENCE ON EVOLUTIONARY COMPUTATION, 2006., 2006. **Anais...** [S.l.: s.n.], 2006. p.1157–1163.

FRANZIN, A.; STüTZLE, T. Revisiting simulated annealing: A component-based analysis. **Computers & Operations Research**, [S.l.], v.104, p.191 – 206, 2019.

GAJSKI, D. D.; ABDI, S.; GERSTLAUER, A.; SCHIRNER, G. **Embedded system design**: modeling, synthesis and verification. [S.l.]: Springer Science & Business Media, 2009.

GENDREAU, M.; POTVIN, J.-Y. (Ed.). **Handbook of Metaheuristics**. 2.ed. New York, NY, USA: Springer Science+Business Media, 2010. v.146.

GLAß, M.; TEICH, J.; LUKASIEWYCZ, M.; REIMANN, F. Hybrid Optimization Techniques for System-Level Design Space Exploration. In: HA, S.; TEICH, J. (Ed.). **Handbook of Hardware/Software Codesign**. Dordrecht: Springer Netherlands, 2017. p.217–246.

GOMAA, H. **Software modeling and design**: UML, use cases, patterns, and software architectures. [S.l.]: Cambridge University Press, 2011.

GONG, L.; DIESSEL, O. Simulation-Based Functional Verification of Dynamically Reconfigurable Systems. **ACM Trans. Embed. Comput. Syst.**, New York, NY, USA, v.13, n.4, Mar. 2014.

GRIES, M. Methods for evaluating and covering the design space during early design development. **Integration, the VLSI journal**, [S.l.], v.38, n.2, p.131–183, 2004.

GRÜTTNER, K. et al. The COMPLEX reference framework for HW/SW co-design and power management supporting platform-based design-space exploration. **Microprocessors and Microsystems**, [S.l.], v.37, n.8, p.966–980, 2013.

GUESSI, M.; NAKAGAWA, E. Y.; OQUENDO, F.; MALDONADO, J. C. Architectural Description of Embedded Systems: A Systematic Review. In: INTERNATIONAL ACM SIGSOFT SYMPOSIUM ON ARCHITECTING CRITICAL SYSTEMS, 3., 2012, New York, NY, USA. **Proceedings...** ACM, 2012. p.31–40. (ISARCS '12).

HALIM, A. H.; ISMAIL, I.; DAS, S. Performance assessment of the metaheuristic optimization algorithms: an exhaustive review. **Artificial Intelligence Review**, [S.l.], 2020.

HARDKERNEL. **ODROID-XU3**. [S.l.]: Hardkernel Co., 2020. https://www.hardkernel.com/shop/odroid-xu3/.

HENDRIKS, M. et al. A blueprint for system-level performance modeling of software-intensive embedded systems. **International Journal on Software Tools for Technology Transfer**, [S.l.], v.18, n.1, p.21–40, 2016.

HERRERA, F. et al. The COMPLEX methodology for UML/MARTE Modeling and design space exploration of embedded systems. **Journal of Systems Architecture**, [S.l.], v.60, n.1, p.55–78, 2014.

hsafoundation. **Harmonizing the Industry around Heterogeneous Computing**. [S.l.]: HSA Foundation, 2020. http://www.hsafoundation.com/.

HUANG, C.-H.; HSIUNG, P.-A.; SHEN, J.-S. UML-based hardware/software co-design platform for dynamically partially reconfigurable network security systems. **Journal of Systems Architecture**, [S.l.], v.56, n.2–3, p.88–102, 2010.

HUANG, Q. et al. Centrifuge: Evaluating full-system HLS-generated heterogenous-accelerator SoCs using FPGA-Acceleration. In: IEEE/ACM INTERNATIONAL CONFERENCE ON COMPUTER-AIDED DESIGN (ICCAD), 2019., 2019. **Anais...** [S.l.: s.n.], 2019. p.1–8.

INDRUSIAK, L. S.; DZIURZANSKI, P.; SINGH, A. K. **Dynamic Resource Allocation in Embedded, High-Performance and Cloud Computing**. [S.l.]: River Publishers, 2016.

INTEL. **Intel Arria 10 Device Overview**. [S.l.]: Intel Corp., 2018. https://www.intel.com/content/www/us/en/programmable/documentation/sam1403480274650.html.

INTEL. **Intel FPGA SDK for OpenCL Software Technology**. [S.l.]: Intel Corp., 2020. https://www.intel.com/content/www/us/en/software/programmable/sdk-for-opencl/overview.html.

INTEL. **Intel Quartus Prime Pro Edition User Guide: Partial Reconfiguration. UG-20136 | 2020.04.13.** [S.l.]: Intel Corp., 2020. https://www.intel.com/content/www/us/en/programmable/documentation/tnc1513987819990.html.

INTEL. **Intel FPGA SDK for OpenCL Pro Edition: Programming Guide. UG-OCL002 | 2020.04.13.** [S.l.]: Intel Corp., 2020. https://www.intel.com/content/www/us/en/programmable/documentation/mwh1391807965224.html.

JÄÄSKELÄINEN, P. et al. Exploiting Task Parallelism with OpenCL: A Case Study. **Journal of Signal Processing Systems**, [S.l.], v.91, n.1, p.33–46, Jan 2019.

JIA, Z. J. et al. A System-level Infrastructure for Multidimensional MP-SoC Design Space Co-exploration. **ACM Trans. Embed. Comput. Syst.**, New York, NY, USA, v.13, n.1s, p.27:1–27:26, Dec. 2013.

JIANG, K.; ELES, P.; PENG, Z. Power-Aware Design Techniques of Secure Multimode Embedded Systems. **ACM Transactions on Embedded Computing Systems**, New York, NY, USA, v.15, n.1, p.1–29, 2016.

JIANG, Y. et al. Design and Optimization of Multi-clocked Embedded Systems Using Formal Technique. In: JOINT MEETING ON FOUNDATIONS OF SOFTWARE ENGINEERING, 2013., 2013, New York, NY, USA. **Proceedings...** ACM, 2013. p.703–706. (ESEC/FSE 2013).

KAELI, D. R.; MISTRY, P.; SCHAA, D.; ZHANG, D. P. **Heterogeneous computing with OpenCL 2.0**. 3rd.ed. Waltham, MA, USA: Morgan Kaufmann, 2015.

KARANDIKAR, S. et al. FireSim: FPGA-Accelerated Cycle-Exact Scale-Out System Simulation in the Public Cloud. **IEEE Micro**, Los Alamitos, CA, USA, v.39, n.03, p.56–65, may 2019.

KHRONOS. **OpenCL Overview: The open standard for parallel programming of heterogeneous systems**. [S.l.]: The Khronos Group Inc., 2020. https://www.khronos.org/opencl/.

LEITE, M.; WEHRMEISTER, M. A. System-level design based on UML/MARTE for FPGA-based embedded real-time systems. **Design Automation for Embedded Systems**, [S.l.], v.20, n.2, p.127–153, 2016.

LIANG, T.; FENG, L.; SINHA, S.; ZHANG, W. PAAS: A system level simulator for heterogeneous computing architectures. In: INTERNATIONAL CONFERENCE ON FIELD PROGRAMMABLE LOGIC AND APPLICATIONS (FPL), 2017., 2017. **Anais...** [S.l.: s.n.], 2017. p.1–8.

LIGNATI, B. N. et al. Exploiting HLS-Generated Multi-Version Kernels to Improve CPU-FPGA Cloud Systems. In: ASIA AND SOUTH PACIFIC DESIGN AUTOMATION CONFERENCE, 26., 2021, New York, NY, USA. **Proceedings...** Association for Computing Machinery, 2021. p.536–541. (ASPDAC '21).

MAKNI, M.; NIAR, S.; BAKLOUTI, M.; ABID, M. HAPE: A high-level area-power estimation framework for FPGA-based accelerators. **Microprocessors and Microsystems**, [S.l.], v.63, p.11–27, 2018.

MARWEDEL, P. **Embedded system design**. [S.l.]: Springer, 2006. v.1.

MELO, M. et al. A parallel Motion Estimation solution for heterogeneous System on Chip. In: INTEGRATED CIRCUITS AND SYSTEMS DESIGN (SBCCI), 2016 29TH SYMPOSIUM ON, 2016. **Anais...** [S.l.: s.n.], 2016. p.1–6.

MICROSOFT. **Project Catapult**. [S.l.]: Microsoft Research, 2020. `https://www.microsoft.com/en-us/research/project/project-catapult/`.

MIELE, A.; DURELLI, G. C.; SANTAMBROGIO, M. D.; BOLCHINI, C. A System-Level Simulation Framework for Evaluating Resource Management Policies for Heterogeneous System Architectures. In: EUROMICRO CONFERENCE ON DIGITAL SYSTEM DESIGN, 2015., 2015. **Anais...** IEEE, 2015. p.637–644.

MISCHKALLA, F.; HE, D.; MUELLER, W. Closing the gap between UML-based modeling, simulation and synthesis of combined HW/SW systems. In: DESIGN, AUTOMATION & TEST IN EUROPE CONFERENCE & EXHIBITION (DATE), 2010, 2010. **Anais...** [S.l.: s.n.], 2010. p.1201–1206.

MODELSIM. **ModelSim*-Intel FPGA Edition Software**. [S.l.]: Intel Corp., 2020. `https://www.intel.com.br/content/www/br/pt/software/programmable/quartus-prime/model-sim.html`.

MUSLIM, F. B.; MA, L.; ROOZMEH, M.; LAVAGNO, L. Efficient FPGA Implementation of OpenCL High-Performance Computing Applications via High-Level Synthesis. **IEEE Access**, [S.l.], v.5, p.2747–2762, 2017.

NOGUEIRA, B. et al. Multi-objective optimization of multimedia embedded systems using genetic algorithms and stochastic simulation. **Soft Computing**, [S.l.], p.1–18, 2016.

PALERMO, G.; SILVANO, C.; ZACCARIA, V. ReSPIR: A Response Surface-Based Pareto Iterative Refinement for Application-Specific Design Space Exploration. **IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems**, [S.l.], v.28, n.12, p.1816–1829, Dec 2009.

PANERATI, J.; SCIUTO, D.; BELTRAME, G. Optimization Strategies in Design Space Exploration. In: HA, S.; TEICH, J. (Ed.). **Handbook of Hardware/Software Codesign**. Dordrecht: Springer Netherlands, 2017. p.189–216.

PAPADIMITRIOU, K.; DOLLAS, A.; HAUCK, S. Performance of Partial Reconfiguration in FPGA Systems: A Survey and a Cost Model. **ACM Trans. Reconfigurable Technol. Syst.**, New York, NY, USA, v.4, n.4, Dec. 2011.

PATEL, H. D.; SHUKLA, S. K. **Ingredients for Successful System Level Design Methodology**. [S.l.]: Springer, 2008.

PIMENTEL, A. D. Exploring Exploration: A Tutorial Introduction to Embedded Systems Design Space Exploration. **IEEE Design & Test**, [S.l.], v.34, n.1, p.77–90, Feb 2017.

PIMENTEL, A. D.; ERBAS, C.; POLSTRA, S. A systematic approach to exploring embedded system architectures at multiple abstraction levels. **IEEE Transactions on Computers**, [S.l.], v.55, n.2, p.99–112, 2006.

POUCHET, L.-N. **PolyBench/GPU - Implementation of PolyBench codes for GPU processing**. http://web.cse.ohio-state.edu/ pouchet.2/software/polybench/GPU/.

QUADRI, I. R. et al. Targeting reconfigurable FPGA based SoCs using the UML MARTE profile: from high abstraction levels to code generation. **International Journal of Embedded Systems**, [S.l.], v.4, n.3-4, p.204–224, 2010.

RIQUELME, N.; VON LüCKEN, C.; BARAN, B. Performance metrics in multi-objective optimization. In: LATIN AMERICAN COMPUTING CONFERENCE (CLEI), 2015., 2015. **Anais...** [S.l.: s.n.], 2015. p.1–11.

ROGERS, P. Heterogeneous system architecture overview. In: IEEE HOT CHIPS 25 SYMPOSIUM (HCS), 2013., 2013. **Anais...** IEEE, 2013. p.1–41.

RUDOLPH, G. **HYPERVOLUME**. [S.l.]: TU Dortmund, 2020. [Online. Accessed on Jan. 01, 2021], `https://ls11-www.cs.tu-dortmund.de/rudolph/hypervolume/start`.

SCARPAZZA, D. P. et al. Software Simultaneous Multi-Threading, a Technique to Exploit Task-Level Parallelism to Improve Instruction- and Data-Level Parallelism. In: INTEGRATED CIRCUIT AND SYSTEM DESIGN. POWER AND TIMING MODELING, OPTIMIZATION AND SIMULATION, 2006, Berlin, Heidelberg. **Anais...** Springer Berlin Heidelberg, 2006. p.12–23.

SHAO, Y. S. et al. Co-designing accelerators and SoC interfaces using gem5-Aladdin. In: MICROARCHITECTURE (MICRO), 2016 49TH ANNUAL IEEE/ACM INTERNATIONAL SYMPOSIUM ON, 2016. **Anais...** [S.l.: s.n.], 2016. p.1–12.

SHAO, Y. S.; Reagen, B.; Wei, G.; Brooks, D. Aladdin: A pre-RTL, power-performance accelerator simulator enabling large design space exploration of customized architectures. In: ACM/IEEE 41ST INTERNATIONAL SYMPOSIUM ON COMPUTER ARCHITECTURE (ISCA), 2014., 2014. **Anais...** [S.l.: s.n.], 2014. p.97–108.

SHATA, K.; ELTEIR, M. K.; EL-ZOGHABI, A. A. Optimized implementation of OpenCL kernels on FPGAs. **Journal of Systems Architecture**, [S.l.], v.97, p.491 – 505, 2019.

SHIELDS, T. **Obtaining Pareto front for more than 2 objectives**. [S.l.]: stackoverflow, 2015. [Online. Answered in stackoverflow forum in Feb. 24 '15 at 23:09. Accessed on Dec. 30, 2020], `https://stackoverflow.com/questions/28707998/obtaining-pareto-front-for-more-than-2-objectives`.

SIGDEL, K. et al. System-level runtime mapping exploration of reconfigurable architectures. In: IEEE INTERNATIONAL SYMPOSIUM ON PARALLEL DISTRIBUTED PROCESSING, 2009., 2009a. **Anais...** [S.l.: s.n.], 2009a. p.1–8.

SIGDEL, K. et al. rSesame - A generic system-level runtime simulation framework for reconfigurable architectures. In: INTERNATIONAL CONFERENCE ON FIELD-PROGRAMMABLE TECHNOLOGY, 2009., 2009b. **Anais...** [S.l.: s.n.], 2009b. p.460–464.

SIGDEL, K. et al. Evaluation of Runtime Task Mapping Using the RSesame Framework. **International Journal of Reconfigurable Computing**, London, GBR, v.2012, Jan. 2012.

SINAEI, S.; FATEMI, O. Multi-objective algorithms for the application mapping problem in heterogeneous multiprocessor embedded system design. **The Journal of Supercomputing**, [S.l.], p.27, May 2018.

SINGH, A. K. et al. Energy-Efficient Run-Time Mapping and Thread Partitioning of Concurrent OpenCL Applications on CPU-GPU MPSoCs. **ACM Transactions on Embedded Computing Systems**, New York, NY, USA, v.16, n.5s, p.147:1–147:22, Sept. 2017.

SUMAN, B.; KUMAR, P. A survey of simulated annealing as a tool for single and multiobjective optimization. **Journal of the Operational Research Society**, [S.l.], v.57, n.10, p.1143–1160, Oct 2006.

TALBI, E.-G. **Metaheuristics**: from design to implementation. New Jersey, USA: John Wiley & Sons, 2009. v.74.

VIPIN, K.; FAHMY, S. A. FPGA Dynamic and Partial Reconfiguration: A Survey of Architectures, Methods, and Applications. **ACM Comput. Surv.**, New York, NY, USA, v.51, n.4, p.72:1–72:39, July 2018.

VOELTER, M.; RATIU, D.; KOLB, B.; SCHAETZ, B. Mbeddr: Instantiating a language workbench in the embedded software domain. **Automated Software Engineering**, [S.l.], v.20, n.3, p.339–390, 2013.

WAIDYASOORIYA, H. M.; HARIYAMA, M.; UCHIYAMA, K. **Design of FPGA-Based Computing Systems with OpenCL**. [S.l.]: Springer, 2018.

WANG, G.; GONG, W.; DERENZI, B.; KASTNER, R. Exploring Time/Resource Trade-offs by Solving Dual Scheduling Problems with the Ant Colony Optimization. **ACM Trans. Des. Autom. Electron. Syst.**, New York, NY, USA, v.12, n.4, p.46:1–46:20, Sept. 2007.

WANG, H.; HE, S.; YAO, X. Nadir point estimation for many-objective optimization problems based on emphasized critical regions. **Soft Computing**, [S.l.], v.21, n.9, p.2283–2295, 2017.

WANG, R.; SONG, X.; ZHU, J.; GU, M. Formal modeling and synthesis of programmable logic controllers. **Computers in Industry**, [S.l.], v.62, n.1, p.23–31, 2011.

WESSING, S. **HYPERVOLUME: Python Version**. [S.l.]: TU Dortmund, 2020. [Online. Python Code. Accessed on Jan. 01, 2021], `https://ls11-www.cs.tu-dortmund.de/_media/rudolph/hypervolume/hv_python.zip`.

XILINX. **SDAccel: Enabling Hardware-Accelerated Software**. [S.l.]: Xilinx, 2020. `https://www.xilinx.com/products/design-tools/software-zone/sdaccel.html`.

ZHAO, J. et al. Performance Modeling and Directives Optimization for High-Level Synthesis on FPGA. **IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems**, [S.l.], v.39, n.7, p.1428–1441, 2020.

ZHONG, G. et al. Lin-Analyzer: A high-level performance analysis tool for FPGA-based accelerators. In: ACM/EDAC/IEEE DESIGN AUTOMATION CONFERENCE (DAC), 2016., 2016. **Anais...** [S.l.: s.n.], 2016. p.1–6.

ZOHOURI, H. R. High Performance Computing with FPGAs and OpenCL. **arXiv preprint arXiv:1810.09773**, [S.l.], p.130, 2018. PhD Thesis.