HAL
open science

Hardware accelerated simulation and automatic design of heterogeneous architecture

Minh Thanh Cong

To cite this version:

Minh Thanh Cong. Hardware accelerated simulation and automatic design of heterogeneous architecture. Hardware Architecture [cs.AR]. Université de Rennes, 2023. English. NNT : 2023URENS002 . tel-04136213

HAL Id: tel-04136213

https://theses.hal.science/tel-04136213v1

Submitted on 21 Jun 2023

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

COLLEGE DOCTORAL BRETAGNE | MATHS, TELECOMS INFORMATIQUE, SIGNAL SYSTEMES, ELECTRONIQUE

Université de Rennes

THESE DE DOCTORAT DE

L'UNIVERSITE DE RENNES

ECOLE DOCTORALE N° 601
Mathématiques, Télécommunications, Informatique, Signal, Systèmes, Electronique
Spécialité : *Informatique*

Par
« Minh Thanh CONG »

« Hardware Accelerated Simulation and Automatic Design of Heterogeneous Architecture »

Thèse présentée et soutenue à « RENNES », le « 15 Mars 2023 »
Unité de recherche : IRISA / INRIA Rennes – Bretagne Atlantique

Rapporteurs avant soutenance :

Abdoulaye GAMATIÉ	Directeur de Recherche CNRS, LIRMM Montpellier
Roselyne CHOTIN	Maître de Conférence HDR, LIP6 Sorbonne Université

Composition du Jury :

Président :	Daniel CHILLET	Professeur, Université Rennes 1
Examinateurs :	Abdoulaye GAMATIÉ	Directeur de Recherche CNRS, LIRMM Montpellier
	Roselyne CHOTIN	Maître de Conférence HDR, LIP6 Sorbonne Université
	Kevin MARTIN	Maître de Conférence, Université Bretagne Sud
Dir. de thèse :	Steven DERRIEN	Professeur, Université de Rennes 1
Co-dir. de thèse :	François CHAROT	Chargé de Recherche, INRIA Rennes

COLLEGE DOCTORAL BRETAGNE | MATHS, TELECOMS INFORMATIQUE, SIGNAL SYSTEMES, ELECTRONIQUE

Université de Rennes

Titre : Simulation accélérée par matériel et conception automatique d'architectures hétérogènes

Mots clés : Conception d'architecture hétérogène, Simulation, FPGA, RISC-V, Système sur puce, Optimisation d'hyperparamètres

Résumé : La conception de plates-formes de système sur puce hétérogènes est complexe avec de nombreuses combinaisons possibles. La simulation détaillée de différentes solutions est nécessaire pour déterminer le meilleur design. Les environnements de simulation existants (tels que gem5) sont limités car purement logiciels et ne prennent pas en compte les architectures hétérogènes. Pour pallier ces limitations, l'utilisation de composants reprogrammables FPGA pour accélérer la simulation est motivée. Notre travail est divisé en deux parties. La première partie est d'ordre expérimental et a étudié une approche de conception d'architectures hétérogènes en se concentrant sur la simulation de modèles de performance de composants de l'architecture (accélérateurs matériels et cœurs de processeurs) sur FPGA. La seconde partie est méthodologique et concerne un flot pour déterminer la meilleure microarchitecture en termes de rapport performance/consommation d'énergie. Ce flot combine un simulateur logiciel d'architecture et une méthode d'optimisation d'hyperparamètres pour trouver la meilleure combinaison de parallélisme, stratégies de déroulage de boucles et interfaces de mémoire. Les expérimentations ont été menées sur différents problèmes pour déterminer les solutions les plus optimales en termes d'efficacité énergétique.

Title : Hardware accelerated simulation and automatic design of heterogeneous architectures

Keywords : Heterogeneous architecture design, Simulation, FPGA, RISC-V, System-on-Chip, Hyperparmeter optimization

Abstract : The design of heterogeneous system-on-chip platforms is complex with many possible combinations. A detailed simulation of different solutions is necessary to determine the best design. Existing simulation environments (such as gem5) are limited as they are purely software-based and do not take heterogeneous architectures into account. To address these limitations, the use of reprogrammable FPGA components to accelerate simulation is motivated. Our work is divided into two parts. The first part is experimental and studies an approach to designing heterogeneous architectures, focusing on simulating models of architecture components (hardware accelerators and processor cores) on FPGA. The second part is methodological and concerns a flow to determine the best microarchitecture in terms of performance to energy consumption ratio. This flow combines a software architecture simulator and a hyperparameter optimization method to find the best combination of parallelism, loop unrolling strategies, and memory interfaces. Experiments were conducted on different problems to determine the most optimal solutions in terms of energy efficiency.

ACKNOWLEDGMENT

With the assistance of different people, this thesis can become a reality. I would like to give my deepest appreciation to everyone involved.

First of all, I would like to express my sincere gratitude to my supervisor, François CHAROT, researcher at INRIA Rennes, for his unwavering support and guidance throughout my thesis work. His expertise, encouragement and insights have been invaluable in shaping this work. I am deeply thankful for his dedication to my growth as a researcher and his commitment to ensuring the success of my project. This thesis would not have been possible without his invaluable guidance and support. Thank you, François.

I would also like to thank Professor Steven DERRIEN of the University of Rennes 1 for his insightful contribution. His insightful comments and suggestions played a significant role in the final outcome of my doctoral thesis.

I would like to express my sincere appreciation to the members of my committee for their contributions to this thesis. Their insightful comments and constructive criticism have been extremely helpful in refining and elevating the quality of my work. Their expertise and guidance were essential in ensuring that the final product met the standards of excellence and rigor expected in a doctoral thesis.

Furthermore, I am grateful for the support of all permanent staff as well as my colleagues on the TARAN (CAIRN) team. Your support has made my work much easier and more enjoyable.

And finally, I would like to express my heartfelt gratitude to my family, including my wife, my daughter, my son, and my parents. Your unwavering support, encouragement, and belief in me have been the driving force behind my achievements and completion of my thesis. Your love and sacrifice have made this journey much easier and enjoyable. Thank you for always being there for me and for making my life complete.

RÉSUMÉ EN FRANÇAIS

L'architecture des processeurs a évolué au cours des dernières années, des structures homogènes vers des systèmes hétérogènes plus complexes, en profitant de l'évolution de la technologie et des capacités d'intégration permettant ainsi une augmentation du nombre de transistors ; cela s'est entre autres traduit par des circuits combinant accélérateurs matériels dédiés économes en énergie et des cœurs de processeurs généralistes. Ces systèmes sur silicium hétérogènes sont des ordres de grandeur plus efficaces et nécessitent moins d'énergie que les processeurs généralistes. Un des principaux problèmes de ces systèmes hétérogènes est qu'ils sont beaucoup plus difficiles à concevoir et à évaluer. Aussi, des approches de conception efficaces sont nécessaires pour aider à spécifier ces systèmes complexes et en particulier à explorer plus rapidement l'espace de conception.

En ce qui concerne la conception de ces systèmes multi-cœurs hétérogènes, le nombre de combinaisons possibles conduit à un grand espace de conception, avec souvent des compromis subtils. Déterminer la meilleure conception pour une application cible donnée nécessite une simulation détaillée de nombreuses solutions possibles. Des environnements de simulation, tels que gem5, existent et sont couramment utilisés pour réaliser ces simulations. Malheureusement, ils ne sont basés que sur des approches purement logicielles et ne permettent pas une véritable exploration de l'espace de conception. De plus, ils ne prennent pas vraiment en charge les architectures multi-cœurs hétérogènes (accélérateurs matériels et cœurs de processeur). Ces limitations motivent l'utilisation de matériel spécifique pour accélérer la simulation, en particulier les composants FPGA reprogrammables.

Cette thèse apporte une contribution aux méthodes de conception de systèmes sur puces hétérogènes (SoC) ciblant les plateformes à base de composants FPGA en explorant automatiquement ou semi-automatiquement l'espace de conception sur la base de critères puissance-performance au niveau système. L'un de nos objectifs est de pouvoir modéliser les processeurs et les accélérateurs de manière à permettre ensuite de les simuler sur des plates-formes FPGA. Il s'agit aussi de déterminer l'architecture optimale pour une architecture efficiente sans nécessiter de connaissances

d'expert.

Nous avons établi une infrastructure de conception basée sur l'utilisation de trois outils de simulation : Aladdin [105], gem5 [14] et HAsim [89]. Le simulateur d'accélérateur Aladdin fournit un cadre pour modéliser la puissance, les performances, l'activité au niveau cycle des accélérateurs autonomes, réalisant une fonction fixée sans avoir à générer de description matérielle de niveau transfert de registre (RTL). Le simulateur architectural gem5 est un simulateur de système bien connu pour la simulation de cœurs de processeurs configurables et des systèmes de mémoire. Le simulateur FPGA HAsim permet de construire des modèles de simulation de processeurs et des systèmes de mémoire, incluant le support pour les protocoles de cohérence de cache et les modèles d'interconnexion.

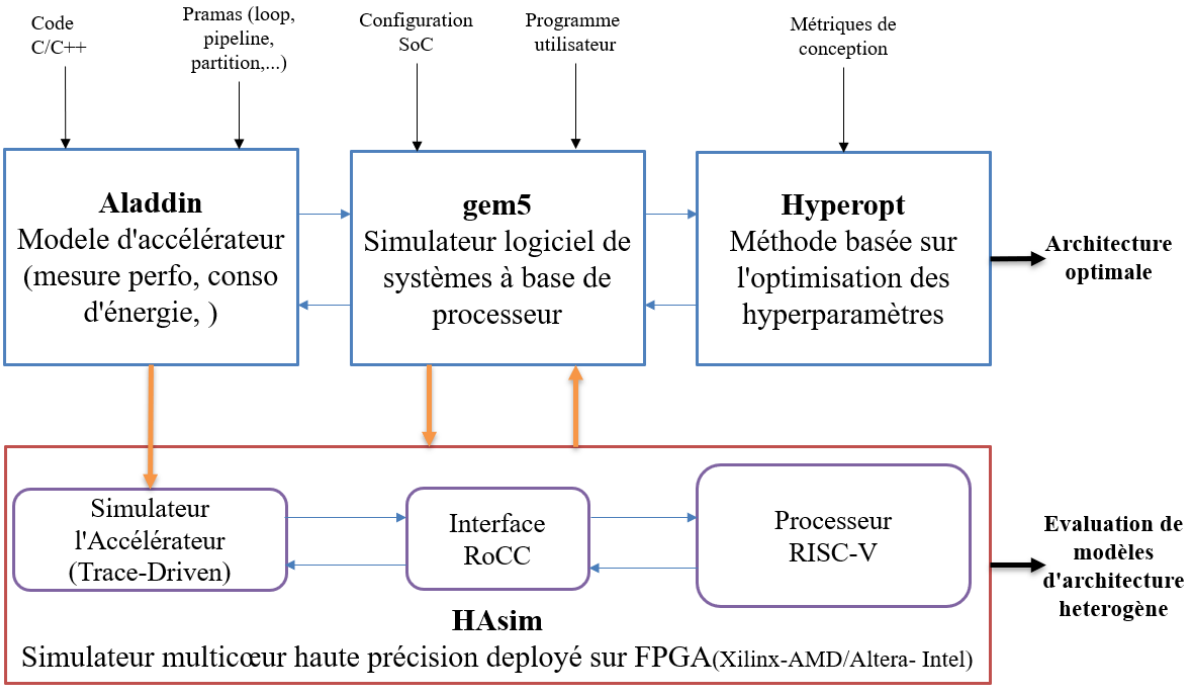


FIGURE 1 – Un aperçu de la méthodologie de conception et des contributions couvertes dans la thèse.

Le haut de la Figure 1 montre un flot basé sur l'utilisation d'un simulateur d'architecture (gem5-Aladdin) et d'une méthode d'optimisation d'hyperparamètre [11] (Hyperopt) dont le but est de rechercher une performance optimale en termes de puissance. Il y a une motivation pour les concepteurs d'outils à aider les architectes, les ingénieurs en logiciel et les développeurs d'algorithmes et ainsi contribuer à améliorer la concep-

tion du matériel. Par conséquent, un flot de conception efficace allant de l'algorithme à l'architecture avec une sélection automatique ou semi-automatique d'une solution optimisée est attractif. Les approches basées sur l'optimisation des hyperparamètres [1] s'avèrent très utiles pour optimiser les fonctions objectives inconnues [104, 12]. En termes de convergence et de qualité des solutions obtenues, il a été montré qu'ils surpassent l'optimisation heuristique.

Comme illustré sur la Figure 1, notre approche repose sur l'utilisation de plusieurs simulateurs, d'algorithmes d'optimisation et d'un flot de conception qui peut être déployé sur une plate-forme matérielle combinant CPU et FPGA. L'environnement prend en entrée des algorithmes décrits en C/C++ ; des paramètres de conception (déroulage de boucles, pipeline de boucles, partitionnement de tableaux, etc.) ; des configurations de SoC ; des programmes utilisateurs ; et des métriques de conception. L'espace de conception est exploré et l'architecture est simulée automatiquement. Nous pouvons choisir de manière flexible d'effectuer la simulation sur le CPU hôte ou sur la plate-forme matérielle CPU/FPGA. Par exemple, si une simulation rapide et précise est requise, elle peut être exécutée sur la plate-forme matérielle CPU-FPGA ; une première évaluation peut être réalisée sur le CPU hôte préalablement au déployement sur la plate-forme matérielle. La prochaine étape de conception est l'exploration de l'espace de conception et l'identification d'une architecture adaptée. A l'issue de l'exploration, le résultat est une architecture appropriée qui satisfait les exigences du concepteur. La plate-forme matérielle CPU-FPGA utilisée pour les expérimentations et le déploiement du simulateur HAsim est une plate-forme d'Intel où les processeurs Xeon et le FPGA sont étroitement couplés.

Le simulateur exploite des traces d'exécution, il utilise le flot Aladdin pour générer et ordonnancer une trace d'exécution sous la forme de graphe de dépendance de données. Le graphe ordonnancé est ensuite traduit dans un format de représentation compact adapté à son interprétation par le module matériel correspondant au modèle du chemin de donnée de l'accélérateur et implémenté dans les FPGA. La stratégie de base est de réutiliser le résultat du simulateur d'accélérateur et de se concentrer sur la simulation de la communication entre les accélérateurs et le reste du système (processeurs, systèmes de mémoire, etc.).

Nous avons choisi de construire des modèles de processeur basés sur le jeu

1. L'optimisation des hyperparamètres est la sélection des paramètres optimaux ou meilleurs pour un algorithme d'apprentissage automatique ou d'apprentissage en profondeur.

d'instructions RISC-V [119], développé à l'Université de Californie à Berkeley. En utilisant l'environnement HAsim, nous avons modélisé différentes versions de processeurs RISC-V (non pipeliné, pipeliné et avec exécution dans le désordre), ceux-ci s'interfacent avec le reste du système composé d'accélérateurs et de mémoire. Afin d'invoquer l'accélérateur depuis le processeur, nous avons ajouté des instructions dédiées au jeu d'instructions RISC-V. L'intégration des blocs matériels est réalisée à travers une interface similaire à celle conçue pour le circuit Rocket [2] (RoCC). Cette interface permet au processeur de communiquer avec le bloc matériel en exécutant des instructions dédiées prises en charge par l'ISA RISC-V.

Toutes les contributions de ce travail s'intègrent dans un environnement pour la conception rapide et efficiente d'une architecture SoC hétérogène. Les travaux peuvent être décomposés en deux contributions principales.

— La première partie du travail présenté dans cette thèse est de nature expérimentale. Elle a porté sur l'étude d'une approche de conception pour les architectures hétérogènes basée sur la conception de modèles de performance pour les composants de l'architecture hétérogène, à savoir les accélérateurs matériels et les cœurs de processeurs. La contribution a porté sur l'expérimentation et l'évaluation des outils de simulation pour ces modèles d'architecture hétérogène sur FPGA. Une méthodologie pour construire des modèles de performance d'accélérateur et un flot de conception ont été proposés.

— La seconde partie du travail est de nature méthodologique. Elle a porté sur l'étude d'un flot pour déterminer, au niveau système, une microarchitecture offrant la meilleure efficacité en termes de rapport performance/consommation d'énergie. Le flot proposé combine deux techniques : l'utilisation d'un simulateur logiciel d'architecture et d'une méthode d'optimisation des hyperparamètres. Cette méthodologie permet de balayer différents types de parallélisme avec différentes stratégies de déroulement de boucles tout en prenant en compte différents types d'interfaces avec les mémoires. Les expériences sur différents problèmes (réseaux de neurones convolutionnels, SoC constitué de plusieurs accélérateurs) ont permis de déterminer les solutions les plus optimales en termes de rapport performance/consommation d'énergie.

2. https ://github.com/chipsalliance/rocket-chip

TABLE OF CONTENTS

Introduction **1**

1 Heterogeneous System-on-Chip achitectures **13**

1.1 The rise of heterogeneous system on chip 14

1.1.1 Technology scaling challenges 14

1.1.2 Trends in heterogeneous architecture 16

1.2 Architectural simulators using FPGAs 18

1.2.1 Simulation wall 18

1.2.2 FPGAs used for simulation instead of prototyping 19

1.2.3 Functional/Timing partitioning simulators 21

1.2.4 FPGA-accelerated microarchitecture simulation projects 23

1.3 Heterogeneous SoC design 26

1.3.1 SoC design Flow 26

1.3.2 Design frameworks for heterogeneous-accelerator SoC 28

1.3.3 Design space exploration 32

1.4 Summary 34

2 FPGA-accelerated simulation of heterogeneous architectures **36**

2.1 Introduction 38

2.2 FPGA-based processor simulation with HAsim 40

2.2.1 HAsim framework overview 42

2.2.2 The LEAP operating system for FPGA-based applications 43

2.2.3 Bluespec system verilog 46

2.3 A case study with the design of RISC-V models within the HAsim framework 49

2.3.1 Semantic of function partition 50

2.3.2 Timing model creation 51

2.3.3 Evaluation results 54

2.3.4 Targeting the Xilinx Virtex-7 FPGA platform 56

2.4 Deploying the HAsim simulator on a Intel CPU-FPGA platform 58
2.4.1 Intel Xeon+FPGA platforms 59
2.4.2 Implementing communication channels support for Intel CPU-FPGA platform 60
2.4.3 Validation 63
2.5 FPGA-Accelerated microarchitecture simulation challenges 65
2.6 Conclusions 70

3 Integration of a pre-RTL accelerator model in the FPGA-based simulator 72
3.1 Introduction 74
3.2 Design flow overview 76
3.3 Accelerator modeling (Pre-RTL accelerator model) 78
3.3.1 DDDG generation and scheduling 78
3.3.2 Scheduled Graph Trace (SGT) generation 79
3.3.3 Flow explanation by an example 81
3.4 Integration of an accelerator model in the HAsim simulator 82
3.5 Performance assessment 86
3.5.1 Case study: Blocked Matrix Multiply accelerator 87
3.5.2 Machsuite benchmarks 89
3.6 Conclusion 90

4 Determining optimal configuration architecture for heterogeneous-accelerator SoCs 91
4.1 Introduction 93
4.2 Design space exploration using Hyperparameter Optimization 95
4.2.1 Synthetic view of a Heterogeneous-Accelerator SoC 95
4.2.2 Design space exploration via Hyperparameter Optimization . . . 96
4.2.3 Hyperopt: Hyperparameter Optimization python library 98
4.3 Design methodology 104
4.3.1 Hyperopt-gem5-Aladdin framework 105
4.3.2 Parallel accelerator exploration 107
4.3.3 Memory coherency models exploration 108
4.3.4 Automatic architectural optimization design flow 109
4.4 Experiments 110
4.4.1 Convolutional Neural Network accelerator in a SoC 111

4.4.2 Multi-context accelerator 113
4.4.3 Coherency interface choice study 116
4.4.4 Hyperopt convergence study 118
4.5 Conclusion 120

Conclusion and future works **122**

Bibliography **125**

Acronyms **139**

LIST OF FIGURES

1 Un aperçu de la méthodologie de conception et des contributions couvertes dans la thèse. ii
2 Energy and area efficiency of different architectures [122]. 2
3 An example of a typical heterogeneous architecture. 3
4 A die photo shows the makeup of several common mobile processors (SoC). The blue boxes are CPU cores, and the red boxes are GPU cores. Most of the area of the processors is not the CPU and GPU blocks, which are taken up by application-specific accelerators. The original die photos are from AnandTech [47], ChipRebel [4], TechInsights [99]. 4
5 Frameworks of the methodology together with covered contributions. . . 8

1.1 Transistor scaling challenges with the evolution of computer systems. 14
1.2 Trends in heterogeneous architecture view. 17
1.3 The use of FPGA in the circuit design flow. 20
1.4 A partitioned simulator is divided into two partitions: functional and timing. 22
1.5 SoC design flow overview [54]. 26

2.1 HAsim Framework Overview. 43
2.2 The HAsim simulator is based on the LEAP Virtual Platform [65]. 46
2.3 A Counter expressed in Bluespec. 47
2.4 Design flows target FPGA platforms using Bluespec System Verilog. 49
2.5 An example of three different timing models operating on the same instruction set. . . 52
2.6 Target processors and their simulator implementation. 53
2.7 Evaluating the performance of the simulators with target processsor models. 55
2.8 FPGA-based platform overall setup. 57
2.9 An overview of the architecture and hardware of Intel Arria systems. The "Green Region" identifies the portion of the FPGA that may be reconfigured in user space during runtime. The "Blue Region" describes the FPGA's static soft core (Intel API). It makes the CCI-P interface accessible to the AFU. 59
2.10 Quick Assist (QA) driver with CCI-P and OPAE. 61

2.11 An overview of the leap environment includes the QA driver modules for the Intel CPU-FPGA platform. 62
2.12 Set of FPGA-based platform development tools. 66
2.13 An overview of modules to develop functional and timing partitions in in-order pipelined processor models. 68

3.1 Illustration of the generic design flow. 77
3.2 Actions taken by the timing accelerator for the SGT format. 80
3.3 An example of an accelerator model with a factor of 2 loop iteration parallelism, partitioning factor 2, and without loop pipelining. 81
3.4 The flow designed to generate a customized accelerator and its simulation models. 83
3.5 An overview of the communications, including the rocket core and accelerator. 84
3.6 Custom instruction format. 85
3.7 Structure of the proposed simulation platform. 86
3.8 The pseudocode of the blocked GEMM algorithm. 87
3.9 Blocked GEMM evaluation. 88
3.10 Performance validation under different architectures. 89

4.1 View of a typical heterogeneous-accelerator SoC. 95
4.2 The pseudo-code of generic Sequential Model-Based Optimization [9]. . 97
4.3 Adaptive-TPE algorithm: a tuning parameters technique for improving TPE. 101
4.4 An example of the TPE procedure with six loop iteration parallelism factors and its Energy-Delay-Product. 102
4.5 An example of a Hyperopt configuration with six loop iteration parallelism factors. 103
4.6 Overview of our generic design flow using the hyperparameter optimization-based method. 105
4.7 Convolutional layer operation of a CNN. 108
4.8 EDP improvement for CNN workloads. 114
4.9 Radar charts of architecture configurations with optimal EDP. 116
4.10 EDP improvement for coherency interface. 117
4.11 The convergence of parallel exploration in LeNet-5 workload. 119

4.12 The convergence of the coherency interface experiment. 120
*

LIST OF TABLES

2.1 Comparison of Research Accelerator for Multiple Processors (RAMP) projects. 41
2.2 Functional partition operations. 51
2.3 Synthesis results for a Virtex-7 VC707 FPGA platform. 57
2.4 Results of experimental application synthesis targeting Arria 10 FPGA. . 64
2.5 CPU-FPGA communication channel validated in Intel Xeon+FPGA. . . . 64
2.6 Lines of Bluespec code to implement the simulation models. 69

4.1 gem5-Aladdin SoC Architecture Configuration. 111
4.2 Unrolling factors for CNN-Workloads(M,N,K,S) (loop_m,loop_n,loop_r,loop_c,loop_i,loop_j). 113
4.3 SoC-Accelerators Design Space, where (M,N,K,S) represents four parameters at the CONV layer C1/C3 ,and x::y::z denotes a set of values from x to z by a stepping factor of y. 115
4.4 Energy Delay Product improvements of the multi-context architecture over the most optimal configuration for each CNN-workload. 115
4.5 Accelerated-workloads in a SoC. 116

*

INTRODUCTION

The desire for improved computing speed is continuous in the computer architecture design community. In addition, artificial intelligence and data science are among the fastest growing technologies that need an increase in hardware's computation power. Self-driving vehicles, 5G communication, video monitoring, and analytics are examples of highly demanding applications. In an effort to support these applications on vehicles and other mobile devices, architects and researchers focus on embedded high-performance computing architectures to solve these problems effectively and quickly.

Designers have already proposed solutions based on processor core duplication to address significant computational demands and have pushed the number of cores on a chip as high as a thousand [15]. This idea led to the evolution of replacing single-core designs with multi-core [3] and many-core [4] architectures to perform several tasks concurrently and continue increasing the hardware performance. However, multi/many-core chips have billions of transistors, which cannot all be activated or switched on simultaneously at high frequencies (utilization wall) [117]. The silicon parts powered off to prevent overheating (referred to as dark silicon) [42] will grow exponentially with each new technology generation. Thus, microprocessor performance gains are slowing down due to modern applications' power and growing computation demands.

In this context, computer architects are moving toward specialization, with designs trading off dark silicon for a collection of customized hardware. As a result, heterogeneous architectures (HAs) combining processor cores with specialized hardware (accelerators) has received increasing interest in recent years. By implementing particular functions in hardware, HAs provide increased power efficiency [5]. Hardware accelerators in specialized datapaths and memory management promise to support highly demanding workloads while increasing performance and energy efficiency. Figure 2 shows the energy efficiency and area comparison between three basic architectural

3. A multicore processor is typically made up of two or more independent processor cores on the same silicon.
4. A many-core processor typically refers to devices with dozens or hundreds of cores.
5. Operations per second per watt.

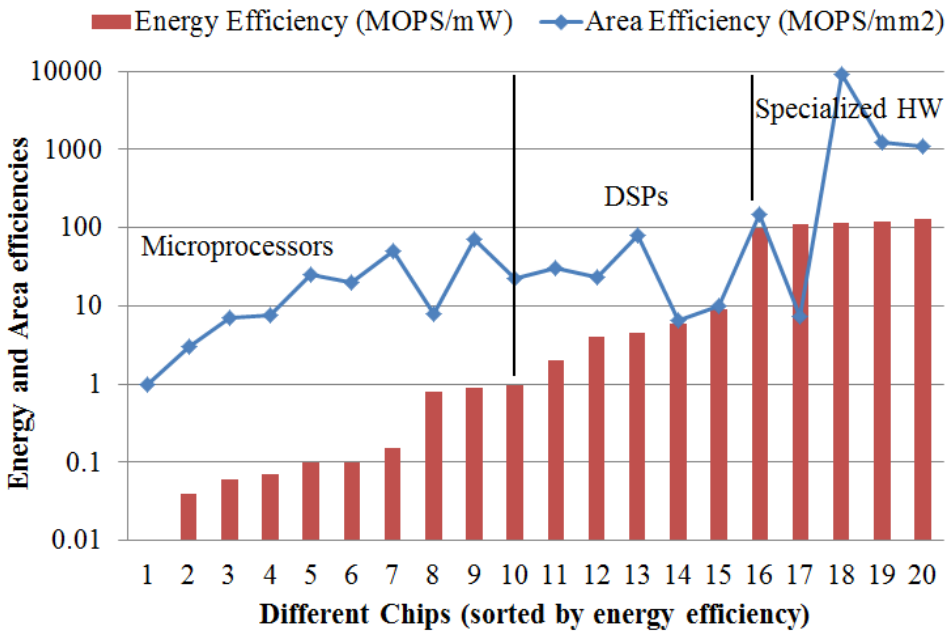


Figure 2 – Energy and area efficiency of different architectures [122].

categories: microprocessors, digital signal processing (DSP), and application-specific hardware accelerators. The data came from 20 distinct chips that were first presented at the International Solid State Circuits Conference (ISSCC) between 1998 and 2002 [122]. When compared to microprocessors (general-purpose CPUs), customized processors such as DSPs perform 10 to 100 times better in terms of energy efficiency, while specialized hardware accelerators perform 1000 times better. Area efficiencies follow a similar trend, with the exception of a few cases when the targeted application specifies a low clock frequency.

Heterogeneous Systems on Chip (SoC)

As a result of this trend, heterogeneous architectures, which include processor cores and multiple hardware accelerators, have emerged as the most important computing platform in a wide range of applications, from embedded systems to data centers [76, 85, 97]. In this study, we focus on accelerators and processors with shared interfaces in heterogeneous architectures. Figure 3 illustrates an example of a typical heterogeneous architecture that trades dark general-purpose cores for a collection of

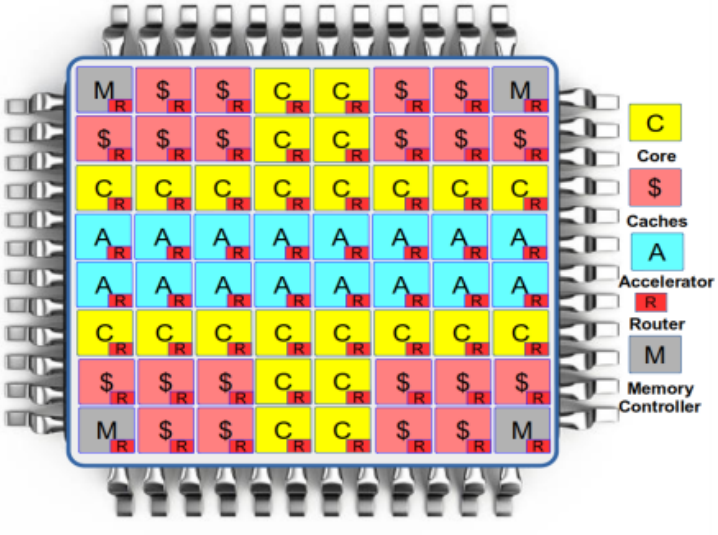


Figure 3 – An example of a typical heterogeneous-accelerator architecture.

customized hardware but transiently powered accelerators [28, 30, 74]. It includes a number of processor cores and many specialized accelerators. Each accelerator comprises several dedicated datapaths that implement parts or all of an algorithm in a specific application domain. Each accelerator has a local memory (scratchpad memory or private cache) to speed up data transfer and thus be able to achieve high performance. The SoC architecture also includes the cache and coherent memory controllers shared by both processor cores and accelerators. At the system-level, the hardware blocks are connected by routers, which represent a customized network-on-chip.

Heterogeneous architecture system-on-chip can be seen in mobile computing areas with, for example, Qualcomm, Apple's A-series, Samsung Exynos, Nvidia Tegra, Texas Instruments OMAP, and HiSilicon Kirin processors. When we take a look at the die photographs in Figure 4, we can see that the blocks that make up the central processing unit (CPU) and the graphics processing unit (GPU) do not take up the majority of the processor's area. According to the findings of Shao et al. [107], the number of specialized hardware blocks that have been implemented throughout all five generations of Apple SoCs has been steadily increasing over the last 10 years. In addition, Intel developed a program called HARP [103] that combines Xeon processor cores with FPGA fabric. This program suggested that the FPGA fabric may be integrated directly onto the chip, which would allow for the implementation of specialized accelerators

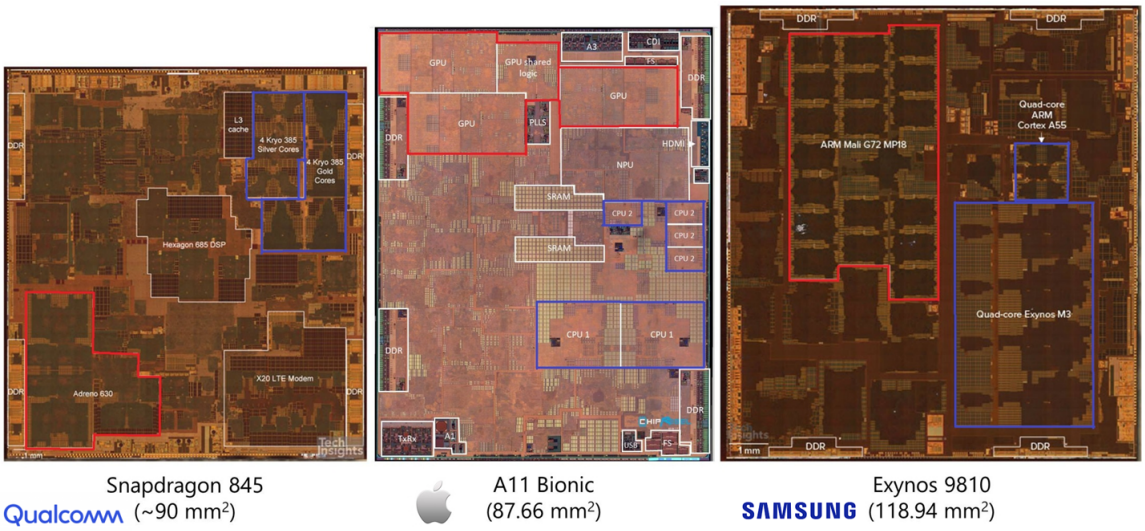


Figure 4 – A die photo shows the makeup of several common mobile processors (SoC). The blue boxes are CPU cores, and the red boxes are GPU cores. Most of the area of the processors is not the CPU and GPU blocks, which are taken up by application-specific accelerators. The original die photos are from AnandTech [47], ChipRebel [4], TechInsights [99].

on a server-class data center CPU. It is clear that more and more chip area is being used by application-specific accelerators. Instead of using a general-purpose CPU to do the same work, specialized accelerators provide much higher performance while using significantly less power.

Design challenges

Unfortunately, the main problem with these heterogeneous systems is that they are much more challenging to design and evaluate. This is especially significant at a time when customers want more powerful technological products in ever-shorter time periods. Therefore, new design methodologies are required to forecast and validate these complex systems more rapidly during the early design phase. The research carried out in this thesis focuses on these topics. Each component in a heterogeneous system has many possible configurations. For example, the design of accelerators can choose how many computing units run concurrently[6], the direct memory access (DMA) or cache system, cache bandwidth, and cache size when integrating with the memory

6. To expose algorithmic parallelism level.

hierarchy. They also drive to choose how a processor can communicate with the accelerator when integrated onto the same silicon die. The overall purpose is to configure specialized accelerators, processors, memory structures, and others to maximize the power-performance efficiency of a given application.

It is obvious that most designers nowadays want to add more and more components to their systems. As a consequence of the many design combinations, there is a huge amount of design space available. Exploring the design space of heterogeneity at the system level requires a significant amount of time, effort, and knowledge. It is essential to rapidly explore and determine an optimal architecture to save cost and design time. Nevertheless, designing heterogeneous systems is still in its early stages and lacks quick and efficient design tools. To tackle the challenge of designing optimal heterogeneous systems, we propose a framework that enables simulation and design exploration rapidly. Our framework relies on an optimization algorithm built on top of the architectural simulation framework.

Simulation techniques

Determining which architecture is best requires a detailed simulation of many different possible solutions. Simulation techniques enable the prediction of many various features without the explicit need to build the system itself. The most commonly predicted computer system features are performance, latency, area, and energy/power consumption. For example, evaluating the impact of the memory system on accelerators is done quickly by changing some of memory configurations (cache size, cache bandwidth, cache associativity, cache line size, etc.) and simulating with a variety of benchmarks. Today, the simulator has gone a long way in validating computer architecture research, and it can rapidly evaluate points in design space to save design time.

However, the performance of computer simulation is decreasing over time. Computer system simulators are typically implemented in software due to the need for flexibility and accuracy. Many simulators used commercially and academically for cycle-accurate or approximate simulation are designed around single-threaded discrete event simulation, with SystemC, Simics, and gem5 based simulation [69], gem5 [14] being one of the most popular tools. Unfortunately, given the rapid growth of their complexity over time, software simulators are becoming slower and slower. A fast and cost-

effective alternative to software is to employ hardware that directly matches the hardware level parallelism required in an accurate computer system simulation [113]. The significant capabilities and flexibility of field programmable gate array (FPGA) make them an ideal vehicle for accelerating and addressing the challenge of computer system simulation. Based on these considerations, a model of heterogeneous accelerator architectures targeting FPGAs toward speeding up the simulation needs to be studied.

Instead of looking for a new design methodology for heterogeneous SoC architectures, we propose to study and develop a design framework based on the existing architecture simulators. The biggest advantage of reusing methodologies is that it saves effort in developing a reliable design framework. The simulation of a particular heterogeneous architecture is built from models of hardware components (processor cores, domain-specific accelerators, memory hierarchies, interconnects, etc.). Detailed models of these components can be specified as hardware description language modules, which can be reused by different architectures.

Design space exploration

Exploring the SoC design space is a challenging task that requires expert hardware architects. Designers have to deal with many design issues and choices that require in-depth knowledge of specific scientific fields [27, 19, 52]. Besides, there is a lack of knowledge on utilizing efficient resources for different application domains, and designers have to explore a huge space to find an optimal architecture. This design space contains architectural design configurations regarding the number of accelerators, their integration with the memory hierarchy, the computational parallelism, and the on-chip interfacing. For example, when designing efficient convolutional neural network (CNN) SoC accelerators, the designer can play with the loop unrolling factor (computational parallelism), interfacing with different memory coherency (data reuse), memory bandwidth, and cache sizing to meet power and performance constraints.

The designer is often constrained by the heuristic design process, which increases design time and costs. There is a motivation for tool designers to help architects, software engineers, and algorithm developers improve hardware design. Therefore, an efficient design flow from algorithms to architectures with automatic or semi-automatic selection of an optimized solution is attractive. Approaches based on hyperparameter [7]

7. They are values/weights that determine the learning process of an algorithm.

optimization [8] proves to be very useful in optimizing unknown objective functions, as stated in works presented in [104, 12]. They are more powerful than heuristic optimization in terms of convergence and quality of obtained solutions.

We are now aware of the challenges that come with the process of heterogeneous SOC designs using software simulators, as well as the challenges that come with exploring the design space. The research motivation related to these problems will be developed later in this manuscript.

Overview of results

This dissertation contributes to SoC design methods targeting FPGA platforms by automatically exploiting a power-performance design space at the system level. One of our goals is to model processors and accelerators so they can be simulated on FPGAs. The other is to determine the optimal architecture for an efficient architecture without requiring expert knowledge.

We established a design infrastructure based on the use of three simulation tools: Aladdin [105], gem5 [14], and HAsim [89]. The Aladdin accelerator simulator provides a framework for modeling the power, performance, area, and cycle-level activity of standalone, fixed-function accelerators without the need to generate RTL. The gem5 architectural simulator is a well-known system simulator with configurable CPU cores and memory systems. The HAsim FPGA-based simulator builds processor timing models and memory systems, including support for cache coherence protocols and interconnect models.

As illustrated in Figure 5, our approach relies on the use of simulators, optimization algorithms, and design flows that can be deployed on a CPU-FPGA platform. The framework takes inputs from: algorithms in C/C++; design parameters (loop unrolling, loop pipelining, array partitioning, etc.); SoC configurations; user programs; and design metrics. The design space is explored, and the architecture is automatically simulated. We can flexibly choose the simulation platform on the host CPU or CPU/FPGA. For instance, if a fast and accurate simulation is required, it can be run on the CPU-FPGA platform; if we need to rapidly evaluate an early design, we can run it on the host CPU. The next step is to explore the design space and identify suitable architecture.

8. Hyperparameter optimization is the selection of optimum or best parameters for a machine learning or deep learning algorithm.

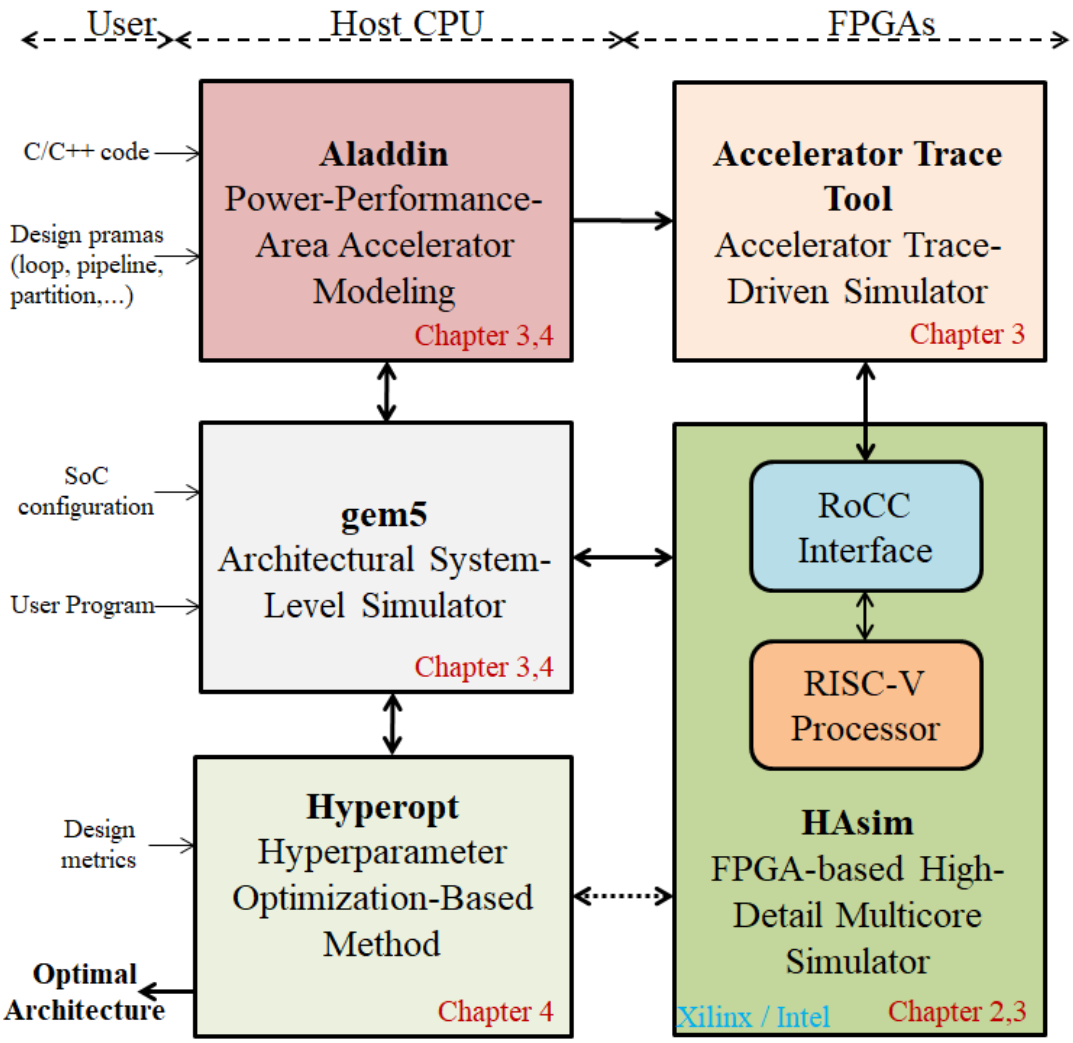


Figure 5 – Frameworks of the methodology together with covered contributions.

After exploration, the result is an appropriate architecture that satisfies the designer's requirements.

To the upper right of Figure 5, the accelerator trace-driven simulator uses the Aladdin flow to generate and schedule an execution trace as a data dependence graph. The scheduled graph is then translated into a compact representation format suited to its interpretation by the hardware module corresponding to the accelerator datapath model and implemented in the FPGAs. The basic strategy is to reuse the accelerator simulator's scheduling and focus on simulating communication between accelerators and the rest of the system (processors, memory systems, etc.).

At the bottom right corner of Figure 5, we chose to build processor models based on the RISC-V [9] instruction set [119], originally developed in the Computer Science Division department at the University of California, Berkeley. Using the HAsim framework, we modeled unpipelined, in-order-pipelined and out-of-order RISC-V processor interfacing accelerator and memory systems. In order to invoke the accelerator from the processor, we added custom instructions to the RISC-V instruction set.

On the left hand side of Figure 5, we have a flow that uses an architecture simulator (gem5-Aladdin) and a hyperparameter optimization method (Hyperopt) to find an optimal power-performance.

Contributions

This thesis focuses on heterogeneous SoC architectures due to their ability to deliver higher performance under the same power budget. The FPGA platforms will be used to simulate specialized hardware accelerators and processor core models. We explored the power-performance of heterogeneous systems and determined the optimal configuration using a hyperparameter optimization algorithm. As a result, we built a fast and efficient design framework for designing heterogeneous SoC architectures.

All of the contributions of this work fit into a framework for fast and effective design of heterogenous SoC architecture. These contributions are presented in detail in the three main chapters and illustrated in Figure 5:

9. RISC-V is a new instruction set architecture (ISA) that was originally designed to support computer architecture research and education, for which it could become a standard open architecture for industry implementations.

Chapter 2

— We present the design of RISC-V processor models within the HAsim framework. The potential of the HAsim simulator for the simulation of heterogenous architectural systems is explored through a case study. By running on top of the virtual platform infrastructure known as LEAP and utilizing the bluespec system verilog (BSV) high-level synthesis language, HAsim is able to reduce the development effort on FPGA.

— In an effort to deploy the FPGA-based simulator on an Intel CPU-FPGA platform, we have developed a communication channel that enables FIFO-managed bidirectional data exchange. This implementation also enables the system memory interface for read and write requests.

Chapter 3

— By integrating a pre-RTL accelerator model into the HAsim simulator, we provide a method for designing application-specific heterogeneous systems. The application is profiled by the dynamic execution trace and is used to construct a data flow model of the accelerator. The architecture models are deployed on an FPGA simulator. As a result, the design of a multiple-core, multiple-accelerator architecture is made easier by automating the design process.

Chapter 4

— An automatic architectural optimization design flow for determining, at the system-level, the microarchitecture with the best efficiency in terms of performance-power ratio is presented. In our work, we employ the hyperparameter optimization library Hyperopt to select the most optimal architecture. Before starting the optimization, the search space, design parameters, and stop criteria have to be defined.

— A case study allowed us identifying the most energy efficient architecture for a convolutional neural network (CNN). We showed that the solution obtained achieves a 2x to 4x improvement in energy-delay-product (EDP) compared to an architecture without parallelism. Furthermore this solution is more efficient than commonly implemented architectures (Systolic, 2D-mapping, and Tiling).

— To demonstrate the efficiency of the heterogeneous SoC design approach, we determined the optimal architecture, including its coherency interface, for a complex SoC made up of six common accelerated-workloads. Three possible coherency models are considered: a software-managed direct memory access (DMA), a shared last level cache (LLCcoherent), and a fully-coherent cache. Our framework allowed us to determine that a hybrid interface appears to be the most efficient; it achieves 22% and 12% improvement in EDP compared to just only using non-coherent and only LLC-coherent models, respectively.

Organization of the manuscript

Chapter 1 provides the state of the art, enabling us to take a comprehensive approach to the topics discussed in this thesis. The rise of heterogeneous systems on chips is defined first, followed by the design methodology relevant to this type of architecture. Following that, we present what exists to accelerate computer architecture simulation by using FPGA. Finally, this chapter illustrates the approach to heterogeneous SoC design, including the flow, existing development platforms, and design space exploration via optimization.

In chapter 2, we mainly focus on FPGA-based simulation, specifically building experiments to explore the capabilities of the HAsim simulator. We introduce LEAP and its simulator, as well as a case study on the building of RISC-V processor models. We also present a contribution that implements communication channels for simulation on the Intel CPU-FPGA platform. Finally, we describe the barriers to FPGA-accelerated microarchitecture simulation development.

Chapter 3 proposes a method for designing application-specific heterogeneous systems by integrating the pre-RTL accelerator model into the FPGA-based simulator. We demonstrate the design flow and proposed design methodology. The accelerator modeling methodology, which is based on the Aladdin simulator, is then presented. We will then show how to simulate heterogeneous systems with HAsim by combining the accelerator and processor models. Finally, MachSuite benchmark experiments are presented and discussed.

In chapter 4, we use gem5-Aladdin and the hyperparameter optimization (Hyperopt) library to explore the design space of power-performance heterogeneous SoCs. This chapter introduces hyperparameter-based optimization methods for unknown objec-

tive functions. We then demonstrate how the Hyperopt-gem5-Aladdin framework and automatic design flow sweep various forms of parallelism with loop unrolling strategies and memory coherency interfaces. Finally, the framework designs a convolutional neural network and a multi-context CNN accelerator. We use the methodology to find the best architecture and coherency interface for a complex SoC with six accelerated workloads.

Finally, the manuscript concludes with an overview of the presented work and a discussion on future research directions.

Publications

Thanh Cong, François Charot. **Designing Application-Specific Heterogeneous Architectures from Performance Models**. MCSoC 2019 - *IEEE 13th International Symposium on Embedded Multicore/Many-core Systems-on-Chip*, Oct 2019, Singapore, Singapore. pp.1-8.

Thanh Cong, François Charot. **Design Space Exploration of Heterogeneous-Accelerator SoCs with Hyperparameter Optimization**. ASP-DAC 2021 - *26th Asia and South Pacific Design Automation Conference*, Jan 2021, Virtual Conference, Japan. pp.1-6.

CHAPTER 1

HETEROGENEOUS SYSTEM-ON-CHIP ACHITECTURES

The objective of the first chapter is to describe the current state of the art in heterogeneous system-on-a-chip (SoC) architecture, which is required for understanding this manuscript. First of all, this chapter discusses the rise of heterogeneous architecture with technological scaling issues and the trend toward SoC. Next, the need for simulation and the problem of all software simulators are addressed. The benefits of using FPGAs for simulation acceleration and simulator parallelization are discussed. Additionally, heterogeneous SoC design is introduced along with the general design flow and current academic and commercial development platforms. Finally, we present the hyperparameter optimization-based method for exploring the design space.

Contents

1.1 The rise of heterogeneous system on chip 14

1.1.1 Technology scaling challenges 14

1.1.2 Trends in heterogeneous architecture 16

1.2 Architectural simulators using FPGAs 18

1.2.1 Simulation wall 18

1.2.2 FPGAs used for simulation instead of prototyping 19

1.2.3 Functional/Timing partitioning simulators 21

1.2.4 FPGA-accelerated microarchitecture simulation projects 23

1.3 Heterogeneous SoC design 26

1.3.1 SoC design Flow 26

1.3.2 Design frameworks for heterogeneous-accelerator SoC 28

1.3.3 Design space exploration 32

1.4 Summary 34

1.1 The rise of heterogeneous system on chip

1.1.1 Technology scaling challenges

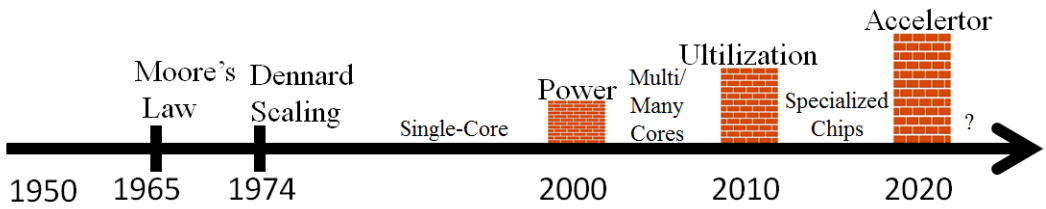


Figure 1.1 – Transistor scaling challenges with the evolution of computer systems.

Gordon Moore published an observation in 1965 that would influence the evolution of computer systems and the semiconductor industry for the next five decades. According to "Moore's Law" the number of transistors that can fit on an integrated circuit will double every two years [81]. This study indicates that we need to make transistors smaller, and Robert Dennard stated in 1974 [35] how to achieve this happen. When voltages are scaled along with transistor dimensions, "Dennard Scaling" means that the device's electric fields remain constant and the majority of device characteristics are preserved. This indicates transistors are becoming smaller and switching faster at the same power density.

Moore's law and Dennard scaling combined provide us with both cost scaling and performance scaling. Consequently, it allowed the performance of microprocessors to increase by 10,000x, a speed that we had never seen before [115]. However, transistor scaling has not fundamentally extended Dennard scaling, which has posed different challenges in recent years. These challenges are considered design walls that have been solved by building new computer architectures.

The power wall

Due to the increasing demands of applications in terms of computing power, processor designers have favored increasing the clock frequencies of the cores of processors. Equation 1.1 describes how energy (E) and power consumption (P) depend on the activity factor (α), the effective capacitance of the transistor (C), the voltage level (V), the operating clock frequency (f), and the delay time (t) [116]. The activity factor

is the probability that the node of the circuit changes from 0 to 1, which is the only time the circuit uses power. The activity factor is affected by the decisions made regarding logic and architecture. The size of the transitor influences the capacitance. When transistors are small, the gate capacitance and the diffusion capacitance are also small. The voltage level has decreased because we need to maintain the same amount of power while simultaneously increasing the clock frequency. When the voltage level is below its threshold, the circuit delay will increase, and the increase in leakage energy will exceed any decrease in switching energy [36, 61].

Dependencies of energy and power consumption

$$E = P * t = \alpha * C * V^2 * f * t \tag{1.1}$$

The power wall forced an increase in processor frequency, which led to excessive power consumption and heat dissipation limitations of cooling systems [96]. Due to the continuous increase in transistor density, transistors are unable to switch faster. As a result, computer designers have embraced multi/many-core architectures. These architectures use a large number of single cores operating at lower frequencies and increase the aggregated performance of the entire chip through thread-level parallelism.

The utilization wall

Starting in the early 2000s, multi-core and many core processors emerged as leading solutions to the power restrictions imposed by voltage scaling. The semiconductor industry has already started duplicating processor cores and has succeeded in increasing the number of cores that may be included on a single chip to up to a thousand [15]. This strategy, however, only postpones the beginning of the power scalability issue. The number of cores in these multi-core and many-core circuits capable of actively switching at full speed while remaining within the chip's power budget will decrease (utilization wall). The remaining silicon, which is left unpowered and is often referred to as "dark silicon," will continue to increase at an exponential rate with each new generation of technology [42]. Moreover, Amdahl's Law [1] demonstrates that the total acceleration is always strongly constrained by the sequential element of the program. This is the case even when the overall speed has increased. Due to power and parallelism constraints, microprocessor performance has hit a wall.

To overcome the utilization wall, heterogeneous architectures combining processor cores and hardware accelerators [28, 73] have been proposed that trade general-purpose dark cores for a collection of specialized but transiently powered accelerators. Silicon customization allows us to construct accelerator-based architectures that efficiently use just the mission-critical transistors for a computation [114]. Compared to performing the same task on a general-purpose CPU, architectures with custom accelerators are orders of magnitude faster and consume significantly less power.

The accelerator wall

In recent years, specialized architectures have enabled significant performance improvements. This raises expectations that growing computing needs will be met when Moore's Law scaling reaches its limit. Fuchs and Wentzlaff predicted that, in 2019, chip specialization would hit an accelerator wall [49]. Based on an analysis of more than 1,000 device datasheets, they determined how current accelerators depend on CMOS scaling. Additionally, case studies were examined to see how computing capabilities scale with a given budget in various applications and chip platforms (e.g., GPUs, FPGAs, and ASICs). Their results showed that the slowdown of CMOS scaling on a chip would limit the accelerator design optimization space, leading to reduced performance improvements and ultimately hitting a wall, much like shrinking transistors.

So if specialized architectures do not continue to provide significant performance improvements once Moore's law scaling breaks down, what will be the solution? First, Wentzlaff suggests that the computer architects shift computations from the non-scaling transistor domain to the still-scaling memory domain [48]. For example, they reused previous computations instead of recomputing them by taking advantage of the fact that the number of flash memory bits on a chip is continuing to increase independently of Moore's Law. Second, despite the prediction hitting the accelerator wall, the specialized architecture is still a good solution for high performance and low power chips. In order to facilitate its advancement and push the technology forward, hardware accelerator optimization needs to continue.

1.1.2 Trends in heterogeneous architecture

The technological scaling crisis encouraged the development of new computer platforms, which often contain numerous diverse components that work together to im-

prove energy efficiency. Multicore chip manufacturing has given us two decades of performance gains, but the desired scaling of performance can't be sustained in the future [77]. Traditional CPU cores, as "multi" as they are, are not as efficient as they should be at running the huge data models that researchers are currently building. This is especially true in intensive applications that use artificial intelligence (AI), such as natural language processing (NLP), image processing, and recommender systems. New architectures that work better with AI and machine learning applications now and in the future are needed.

For example, in gaming, to support increasingly complex image processing and rendering, these accelerators have evolved from microprogrammed processors to fully programmable systems known as graphics processing unit (GPU). They process tasks with a high degree of parallelism. Because of this parallelism, GPU evolved into massively parallel systems, merging them onto a single chip with an increasing number of small cores. Despite their success and widespread deployment in data centers, where better performance can still be achieved, GPU fall far short of the expected energy-efficiency requirements needed for future systems. By connecting CPUs, GPU, and hardware accelerators, specialization and heterogeneity could reduce the efficiency gap [53].

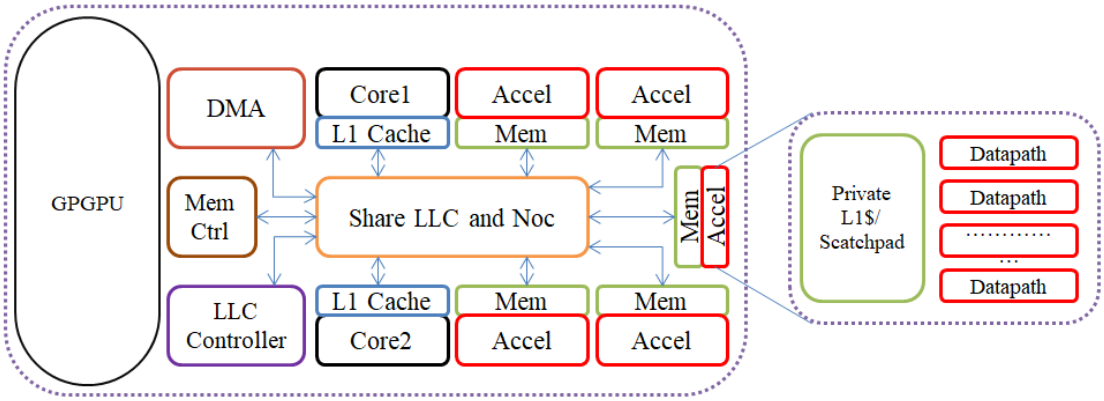


Figure 1.2 – Trends in heterogeneous architecture view.

Most complicated SoCs are now heterogeneous, as seen in Figure 1.2. They are comprised of embedded processors, GPU, and a sea of specialized hardware accelerators, all of which are linked together via shared memory and network on chip (NoC). Heterogeneous SoCs are faster than general-purpose systems and use orders of mag-

nitude less power. The integration of dedicated blocks into SoCs has led to an increase in their performance, despite the fact that the design process has become more complicated and the verification difficulties have increased.

1.2 Architectural simulators using FPGAs

1.2.1 Simulation wall

In computer architecture, simulation is the common and standard way for evaluating the performance of a computer system. There are several reasons for its extensive use. Although analytical models lead to a quick assessment and give a lot of information, they are not accurate enough for many design decisions that an architect must make. One could argue that analytical modeling is useful for making high-level design decisions and figuring out which parts of the huge design space are of interest. However, it is more difficult to analyze modest performance differences across design choices using analytic models. On the other end of the scale, while hardware prototypes are extremely accurate, their development is too time-consuming and expensive. Simulation has the benefit of inexpensive development compared to building hardware prototypes, and it is often more accurate than analytical models.

The simulator is adaptable and easily parameterizable, enabling exploration of the architectural design space, a crucial feature for computer architects designing a microprocessor and researchers analyzing a novel concept. The simulation of computer systems make it possible to predict many behaviors without the explicit need to build the system itself. The characteristic of a computer system that is most frequently predicted is its performance, which is frequently measured in terms of the number of cycles required to execute a set of instructions. Other frequently expected features include energy/power consumption and fault-tolerance reliability.

The introduction of heterogeneous architecture has led to an increasingly diverse set of computer architecture ideas being considered. When traversing the design space to understand the performance of individual design parameters, computer architects often need to run a huge number of simulations. Therefore, they deploy their simulations across a huge cluster of computers. All of these simulations are independent of each other and are often run in parallel. The increase in simulation throughput is related to the quantity of simulation machines in distributed simulation. Indeed, this does not min-

imize the time required to produce a single simulation result. This is a significant issue for simulation runs that take days or weeks. Waiting for these simulations to finish is inefficient and slows down the entire design process.

In real situations, it is best to run simulations that take a few hours or less. Or, to put it another way, it's necessary to quickly achieve a unique simulation result because it may be a crucial discovery that supports research and development. Moreover, industrial and academic architects traditionally use software-based simulators that are cycle-accurate but slow. Murkherjee et al. stated that the slowing down by a factor of 2 relates to the target per year [23]. At the same time as their simulator capabilities have improved, computer designers have reduced their ability to model their next-generation systems. The term "simulation wall" which is accurate and slow, refers to this particular situation [2].

1.2.2 FPGAs used for simulation instead of prototyping

The use of parallelism is a method of trying to speed up individual simulation runs. There are three ways to achieve this [38]. The first is the sampled simulation, in which the sampling units are spread over a cluster of computers. The second technique is parallel simulation, which uses coarse-grain parallelism to transfer a software simulator onto parallel hardware, such as a multicore processor, a cluster of computers, etc. The third is FPGA-accelerated simulation, which uses fine-grained parallelism by mapping a simulator onto field programmable gate array (FPGA) hardware. The advantage of FPGA-accelerated simulation over traditional software simulation is that parallel work on the target architecture can also be performed in parallel on the FPGAs. In addition, FPGAs are less expensive and have a faster turnaround time for new hardware than ASICs. As Moore's law implies that it will be possible to implement an increasing number of components in a single FPGA, we can continue to take advantage of technological developments and design next-generation systems using current technology.

The FPGA is an integrated circuit with an array of programmable logic blocks and a hierarchy of reconfigurable interconnects that allows the linking of logic blocks. After fabrication, the FPGA can be reprogrammed according to the specifications of the application or functionality. This feature distinguishes the FPGA from application specific integrated circuit (ASIC), which is custom-built for specific design tasks. As shown in Figure 1.3,the FPGA has typically been used in different contexts of the computer

architecture design flow:

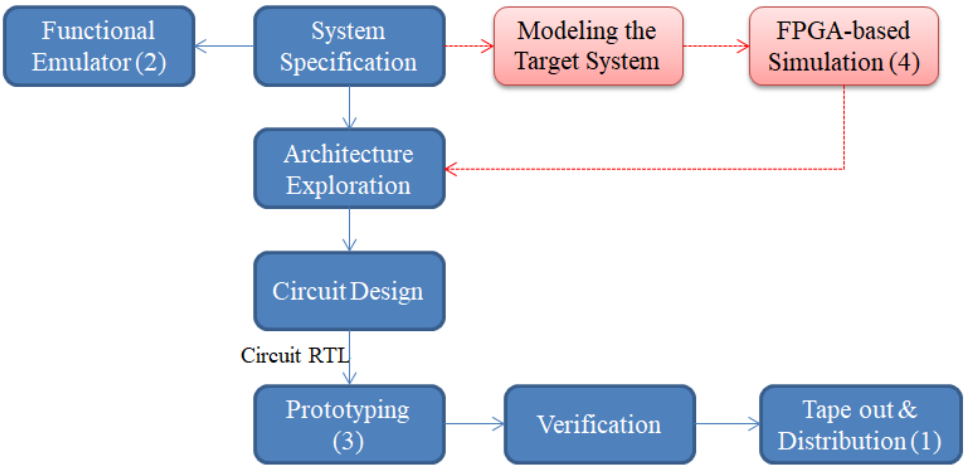


Figure 1.3 – The use of FPGA in the circuit design flow.

— (1) The FPGA is used as an end product that provides reprogrammed hardware. In this case, the RTL circuit description is created using FPGA CAD tools. This usage is for applications that involve video and image processing, high-performance computing, and data centers.

— (2) In function emulation, the FPGA is used to construct a design that implements the functionality of the final system. However, timing information for the individual components is missing. The goal is to develop a functionally accurate emulation with minimal design effort. These designs can use FPGA-specific structures and avoid inefficient FPGA ones.

— (3) The FPGA is used for circuit prototyping, where it helps in verification before the expensive manufacturing step. The target [1] micro-architecture is mapped element by element onto the FPGA and constitutes a prototype. When the work of implementing and integrating the target isn't that different from the manufacturing of the target itself, it's difficult to use FPGA to directly prototype more complex cores and systems.

— (4) FPGA accelerates architecture simulation, the effort focuses on configuring the FPGA into a simulator rather than a prototype. A simulator helps the architect to make good architectural decisions through exploration.

1. The computer system being simulated.

When developing a simulator using FPGAs as the host[2] platform, the objective is to recreate the behavior of the target system with the desired degree of completeness, accuracy, and speed. An FPGA-accelerated simulator can use building components simplifications, such as constant memory latency or implementing only cache tags (and not the data storage). Accuracy is sacrificed to simplify the simulator's implementation. However, accuracy is not necessarily compromised if the simulator produces sufficiently accurate results, even if it does not model every component perfectly.

A designer can combine hardware and software to generate a faster simulator with minimal additional development time and cost compared to a software-only simulator. FPGA-accelerated simulators are hybrid simulators in which FPGAs are used to accelerate specific components of the simulator but not necessarily the entire simulator. Therefore, well-designed FPGA-accelerated simulators are faster than software-only simulators and may even be faster than the physical prototype of the target. They offer greater capability, including full system support, than would otherwise be possible.

1.2.3 Functional/Timing partitioning simulators

To improve performance through parallelization, there are many ways to partition simulators. Separating simulators into functional and timing partitions is one approach that could be taken. The timing partition makes predictions about the performance (and/or power, temperature, dependability, etc.) of the target system, while the functional partition mimics the operation of the target system. Functionality changes little because it is exposed to the entire software stack as a contract through the ISA, but the microarchitecture, which is described in the timing model, changes frequently. Thus, most changes between architectural refinements are in the timing model, while the same functional model can be designed and tested once and then used with only minor changes. As shown in Figure 1.4, there are five basic functional and timing simulator architectures [2] organization. They are monolithic (sometimes called integrated), timing-directed, functional-first, timing-first, and speculative functional-first.

A monolithic simulator integrates the prediction of target functionality and performance into a single block of code. This is a possible implementation and can be considered as prototype, but it is possible that it was built differently or simplified in some way in comparison to the objective. Monolithic simulators are expensive to develop

2. The physical machine on which the simulator runs.

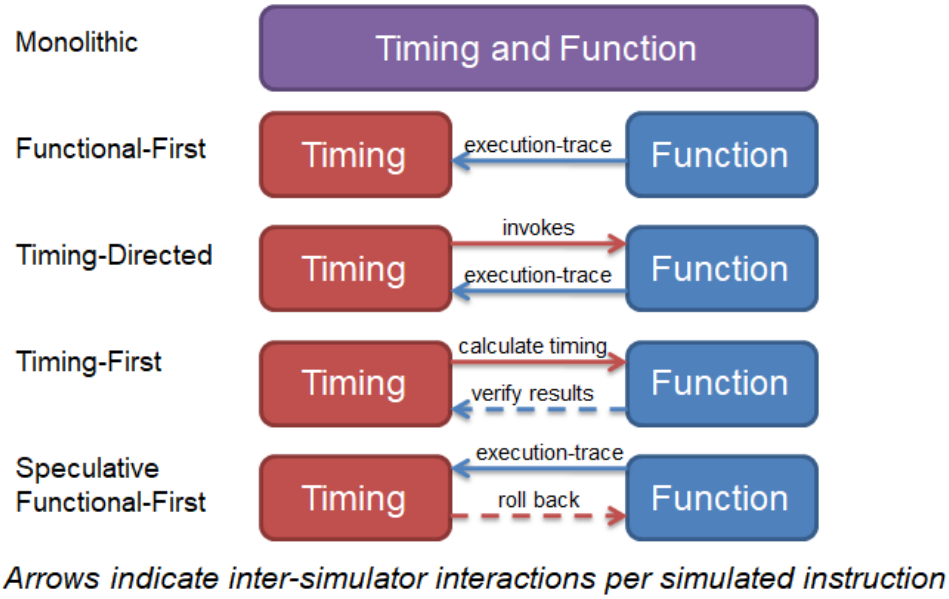


Figure 1.4 – A partitioned simulator is divided into two partitions: functional and timing.

and update, despite the possibility of high precision. Because they are not partitioned into functional and timing components, it becomes challenging to accelerate specific simulator components in the FPGA. Very often, the prototypes are not flexible enough to allow exploration, as they only mimic their final microarchitecture. RAMP Blue [62], RAMP White [3] are two examples of monolithic simulators built on FPGA.

Timing-directed simulators are subdivided into timing models and functional models. The timing model determines the execution of specific functions. It then sends a request to the relevant functional model, asking it to execute the specified operation and provide the result. To be accurate, the timing model must involve a large number of concurrent events that are highly interconnected. Timing and functional models are intimately integrated into a timing-directed simulator, with two-way communications occurring multiple times on each target cycle. The functional model is built separately from the timing models and can be used with multiple timing models. The amount of resources needed to implement the functional partition in the FPGA depends on the complexity of the modeled instruction set. Intel/MIT HAsim [87] and RAMP-Gold [112] are examples of timing-directed simulators that have been built on FPGAs.

Functional-first simulators are built on the presumption that timing has no effect on functionality. Unlike a time-driven simulator, functionality is performed before timing.

It produces an instruction trace that includes the opcode, source registers, destination registers, and data addresses. A timing model implemented on an FPGA is comparable to a functional-first simulator. The hardware essentially implements a significant amount of tightly connected concurrent activities. An instruction trace can be created by a simulator, a virtual machine, or a microprocessor configured to generate traces. The functional-first simulation of a 4-way superscalar processor using an FPGA is demonstrated in ReSim [50]. It operates at around 28 MHz, which is significantly quicker than a software-based timing system model.

A timing-first simulator [79] is composed of a performance simulator that implements the desired functionality. It is possible to consider it as a timing-directed simulator or even as a monolithic simulator. It does not have to implement the whole instruction set or peripheral functions, nor does it have to be accurate. The functional simulator is responsible for identifying and fixing any errors or omissions that may occur. To our knowledge, no timing-first simulators have been developed or accelerated on an FPGA. This is possible with an FPGA-based performance model and a software-based functional model. Timing-first simulators are limited by the speed of the slowest component. Such a technique would maintain the constraints of timing-first simulation, such as the inability to simulate alternative memory models and errors induced by any performance simulator mistake or omission.

The speculative functional-first (SFF) simulator is based on the functional first simulator but provides opportunities for parallelization and FPGA speed up while solving the accuracy problem. The functional model of an SFF simulator operates and initially populates the trace without input from the timing model. Instead of assuming that the information is correct, the timing model reads information from the trace as needed. Since the functional model is executable in software, it is possible to extract it from an existing simulator. UT-FAST [23] is an example of a speculative functional-first simulator. It is presented in the section 1.2.4.

1.2.4 FPGA-accelerated microarchitecture simulation projects

In the 1990s and early 2000s, many academics turned to FPGAs as efficient prototyping and emulation tools for ASICs. Additionally, FPGAs would be used as a device for microarchitectural models written in RTL rather than directly implementing an ASIC design. Initiated in 2007, the research accelerator for multiple processors (RAMP)

project [120] focuses on the vast majority of research in this field. The objective of the RAMP project was to build a shared infrastructure for full-system simulation that would be more effective than traditional software simulators for studying thread-parallel machines. Indeed, the FPGA's highly parallel, programmable execution substrate is suitable for the requirements of multi-core simulation. It can provide multiple orders of magnitude speedup over pure software simulators for detailed models, as shown in different papers [112, 87]. Efforts to explore FPGA-accelerated simulation gave rise to several simulators: FAST, ProtoFlex, RAMP Gold, and HAsim. These simulators all employ functional and timing model partitioning, but they differ in how the two partitions are mapped to the execution platform (composed of the FPGA board and the host computer). This has led to different architectures of simulators as classified in [79, 24]: time-directed (RAMP-Gold, HAsim), functional-only (ProtoFlex), and speculative functional-first (FAST):

— FAST [22, 21], developed at the University of Texas, was a cycle-accurate x86 simulator that leveraged a split, CPU-hosted functional model and FPGA-hosted timing model. It uses a functional software emulator, QEMU [7], highly modified to introduce instruction trace generation, checkpointing, and rollback. This emulator feeds the resulting instruction trace into a timing model implemented in a FPGA.

— RAMP-Gold [113], developed at Berkeley, uses a FPGA to perform cycle-accurate simulation of multi-core. It can simulate up to 64 cores on a Xilinx Virtex-5 FPGA. RAMP-Gold comprises two main in-FPGA components: a functional model of the cores, which time multiplexes the multiple instances of the simulated core, sharing memory and cache resources between the models; and a separate timing model, which drives the functional model's scheduler.

— ProtoFlex [26], developed at Carnegie Mellon University, uses a pipelined core simulation engine, called BlueSPARC, in FPGA to simulate up to 16 instances of a SPARC processor model. ProtoFlex supports full system simulation by falling back to a Simics [75] based simulation when the FPGA model cannot simulate some parts of the simulation. It can boot commercial operating systems. ProtoFlex was the first system to introduce hierarchical simulation and host multithreading as techniques for reducing the complexity of simulator development and to virtualize hardware resources [25].

— HAsim [87], developed at MIT and Intel, used timing and functional models on

FPGA. Additionlly, it provided more detailed models of the pipeline and the memory hierarchy. It is presented in more detail section 2.2.1.

The use of FPGAs for microarchitecture simulation was investigated around the same time by other teams unrelated to the RAMP project. DART [118], an FPGA-based NoC simulator, is a good example. It uses multithreading, like many RAMP simulators, but it also uses NoC-specific model abstractions to allow a wide range of model parameters to be changed at runtime.

The RAMP project has led to FPGA accelerated model execution (FAME) techniques [113], which summarize many of the contributions of the FPGA-accelerated simulation work into three dimensions: host decoupling, abstract RTL, and multithreading. RAMP-Gold and HAsim utilize all three techniques:

— When using the host-decoupled FPGA technique, a target simulation cycle is executed over many FPGA cycles. With host decoupling, ASIC structures that don't map well to FPGA fabric can be replaced with structures that are better on FPGA but take more host cycles to execute. This saves FPGA resources and improves host-cycle time.

— Components of an abstract-RTL FPGA-accelerated simulator do not exactly model the implementation RTL. Through abstraction, components of the target can have their complexity reduced, allowing for a reduction in the amount of FPGA resources required.

— A multithreaded FPGA-accelerated technique uses a single physical datapath on an FPGA to mimic many virtual instances of a block or module inside the target. Each virtual instance has a copy of the target state, and the scheduler chooses which virtual instance to simulate during each host cycle. Multithreading increases target mapping efficiency by reusing expensive logic over many target state copies, which can be mapped onto many FPGA BRAMs and registers.

The RAMP project and FPGA-accelerated simulation did not impress the computer architecture community. There are several technical reasons for this, including the absence of a strong open ISA at the time (only SPARC was available), which in part motivated the development of RISC-V. A RISC-V Chisel-to-Verilog simulator [1] converter transforms a simulator written in a new hardware construction language (Chisel) developed by UC Berkeley into Verilog HDL. After that, processor simulation can be run

1. https://riscv.org/wp-content/uploads/2015/01/riscv-chisel-tutorial-bootcamp-jan2015.pdf

directly on FPGA. The ADEPT Lab designed FireSim [60], an open-source, FPGA-based hardware emulation framework hosted in the public cloud, to reduce the cost of performing fast and accurate full-system simulation. Both academics and industry have used FireSim, particularly for evaluating the performance of new microarchitecture features implemented as Rocket Chip extensions. FireSim uses the public cloud, namely Amazon Web Services' Elastic Computer Cluster (EC2), to solve many of FPGA-based simulation's issues by providing elasticity, scalability, and reduced capital expenses.

1.3 Heterogeneous SoC design

1.3.1 SoC design Flow

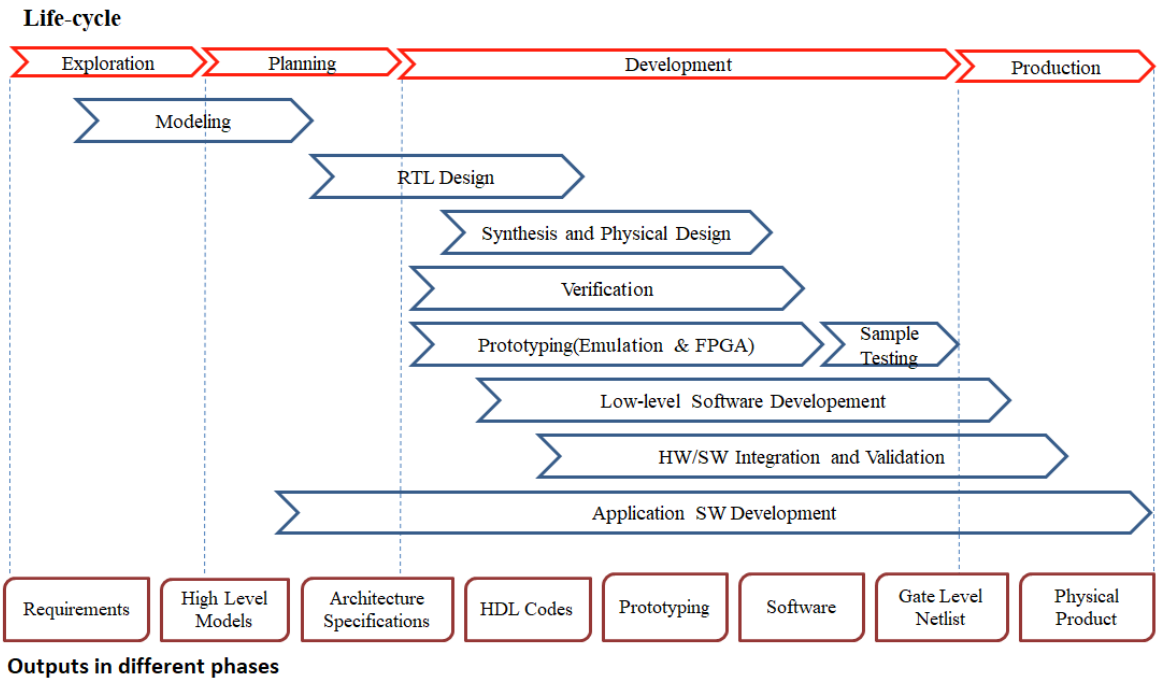


Figure 1.5 – SoC design flow overview [54].

The SoC design flow is not an integrated flow or a single method that produces the chips. Despite the increasing use of computer-aided design (CAD) tools, the standard SoC design cycle is still quite complicated and requires the use of a large number of dedicated tools by expert architects. Figure 1.5 illustrates the design flow for SoC and

shows the process descriptions that provide a common understanding of SoC project operations from exploration to production [54]. It starts with a system design (exploration, planning, and modeling); after that, designers manually build the digital design in register transfer level (RTL) using Verilog or VHDL or use high-level synthesis (HLS) tools. Once all physical designs are complete and validated, the team develops prototype versions and low-level software. Finally, hardware and software are integrated, and application software is developed for the final product. Although the process is iterative, it is usually presented as a single iteration.

Low-level languages such as Verilog and VHDL, which are used in most hardware designs today, require lots of knowledge to create efficient designs. In order to scale with rapid SoC design cycles, the design process must be faster. Hardware engineers must use higher-level programming languages and tools for the next generation of SoC designs. Additionally, higher levels of abstraction in hardware design could make it easier for application programmers to test high-level algorithms in hardware. To increase the degree of abstraction in hardware designs, academics and industry are increasingly focusing on HLS tools that transform a high-level algorithm description into low-level RTL code. It is possible to automate the process of converting functional requirements to RTL using HLS. Specifically, HLS claims to accelerate the design flow using an error-free technique that enables designers to separate the functional behavior definition from the micro-architectural choices for its implementation. Although HLS is necessary to improve the degree of abstraction in SoC architecture, some limitations remain. For instance, HLS tools provide a complex design cycle to create the optimal RTL for a single component but not for system integration.

System-level design has been recommended for years [100, 27, 77] but computer-aided design (CAD) tools still provide little assistance during the exploration and planning stages. Design work often begins with an exploration to understand the requirements and target ASIC technology early to acquire an idea of the feasibility and constraints of the implementation system. A top-down design approach is typically employed, with the construction of a system architecture model that predicts system power, performance, and area. Once the architectural model is complete, it can be evaluated as a simulation model to provide system architecture decisions. The design and validation processes are prolonged due to the difficulties of integrating a growing number of heterogeneous components. My thesis addresses this difficulty and uses commonly used simulation technologies to facilitate the system design of complex sys-

tems. Recently, there have been attempts by industry and academics to assist in the development of SoCs, which are discussed in the following section.

1.3.2 Design frameworks for heterogeneous-accelerator SoC

We are seeing many efforts aimed at proposing novel architectures for a wide range of applications and new tools to design heterogeneous-accelerator SoCs. There are frameworks that can simulate several heterogeneous components, such as specialized accelerators and processors or heterogeneous multi-core architectures. Similar frameworks are also being developed to enable RISC-V processors, which can be implemented in a variety of application domains and are increasing in popularity. There are several projects that can design and generate FPGA prototypes. In an effort to aid in the development of SoCs, researchers and engineers alike have created CAD tools for HLS and programming languages that increase the abstraction level in hardware design.

Simulation of Accelerator-Processor coupling

Co-processors. Several CPU cores operate on a specific problem, and supporting chips have traditionally been referred to as "coprocessors." There is a trend to use coprocessors to offload and accelerate domain-specific applications in order to obtain significant performance improvements and energy/power reductions. For example, the most well-known coprocessor in the history of the PC is the mathematical coprocessor, which later became the floating point unit, or FPU. Technically, an accelerator is a coprocessor, but it has more independence than coprocessors since it is not responsible for completing an entire process; instead, the CPU ignores it except to receive the final results or to determine when the work is complete. Suleyman et al. [101] present a generic methodology for designing domain-specific heterogeneous many-core architectures based on accelerators integrated into simple cores that is comparable to ours. They reveal the steps undertaken to integrate the accelerators into an open source core and use them via custom instructions. They automate custom hardware generation to facilitate design space exploration of heterogeneous architectures. A trace-driven simulation framework for multi-processors using FPGA is presented in [40]. The simulator's input is a specially developed compact execution trace (CET) of the application. The application trace is generated in an architecture agnostic executable format that the

timing model can directly interpret on the FPGA to re-create the original application's execution events.

Accelerator-centric architectures. Simulation platforms adapted to accelerator-centric architectures are proposed in PARADE [27] and gem5-Aladdin [106]. Both provide simulation platforms that enable the exploration of many accelerator designs. PARADE is a simulation platform that can automatically generate a high-level synthesis (HLS) description of the accelerator when the AutoPilot [123] and Synopsys Design Compiler [2] are used. Unlike PARADE, gem5-Aladdin models accelerators from a dataflow representation extracted from the profiling of the dynamic execution of the program, enabling fast design space exploration. The gem5 simulator [14] is then used for the simulation. With such approaches, the designer is faced with the problem of a trade off between accuracy and speed of the simulation. To speed up the simulation, there are approaches where performance models are deployed on FPGA-based platforms, as presented in Chapter 2.

Heterogeneous Systems-on-Chip using RISC-V

Recent platforms are built with a processor-centric perspective, and most of them are based on the RISC-V open-standard instruction set architecture (ISA). Using the Chisel RTL language, the Rocket Chip Generator project creates SoCs with many RISC-V cores linked through a TileLink bus [68]. Celerity [33] uses the Rocket chip's custom co-processor interface (RoCC) to combine five Rocket cores, an array of 496 smaller RISC-V cores, and a binarized neural network (BNN) accelerator developed with HLS it in a 385-million-transistor circuit. Blackparrot [92] is a multicore RISC-V architecture that provides some support for the integration of loosely coupled accelerators. At the moment, it supports fully-coherent and non-coherent caches, out of a total of four that are supported by ESP. As part of our research, we also developed RISC-V-based system-on-chips (SoCs) with support for a number of different cache-coherence options. An adaptive tile-based many-core architecture known as AGILER [59] is presented as a solution for heterogeneous RISC-V-based computers. The suggested architecture is made up of heterogeneous modular multi-/single-core computing tiles that are both adaptive and modular. These tiles support 32-bit and 64-bit RISC-V instruction sets and have various memory hierarchies. In order to provide a high level of system

2. https://www.synopsys.com/implementation-and-signoff/rtl-synthesis-test.html

scalability, the communication between tiles is designed to be based on a scalable network-on-chip architecture. Run-time adaptability is supported by AGILER thanks to a specialized internal reconfiguration manager that allows for dynamic and partial reconfiguration across Xilinx FPGAs.

FPGA-based full-system prototyping

There are several projects with the ability to full-system design and generate FPGA prototypes. Embedded Scalable Platforms (ESP) [93] offers RTL, high-level synthesis (HLS), and machine learning frameworks as three accelerator flows. Each of the three design flows merges into the ESP automated SoC integration flow, which creates the required hardware and software interfaces to quickly enable full-system prototyping on FPGA. The ARA Prototyper [20] is an FPGA-based technique that uses the ZYNQ system to create a sea of accelerators managed by the processor cluster. It has the same objective as in my research, which is to enable quick system-level Design Space Exploration (DSE) with FPGAs. However, the overall architecture is prototyped, restricting the scalability of the desired design, and DSE at several layers of abstraction is difficult to perform. Cosmos [94] leverages both HLS and memory optimization tools to improve exploration of the accelerator design space. Centrifuge [55] is able to generate and evaluate heterogeneous accelerator SoCs by combining HLS with FireSim [60], a FPGA-accelerated simulation platform. All these works provide design frameworks for evaluating accelerators, and it is up to the user to select the optimal one. Unlike these projects, we aim to provide a unified framework for the design, simulation, and optimization of the architecture of accelerator-based SoCs.

High-level synthesis

Currently, low-level languages, such as Verilog and VHDL, are primarily used to program hardware designs. Using low-level languages to create efficient designs takes skill. Even for hardware experts, it's time-consuming. A method of design that is too slow will not scale with SoC design cycles or hardware accelerators. The low-level programming style prevents application programmers with minimal expertise in hardware design from transforming apps into hardware. In addition, the history of software development shows the value of higher abstraction levels to tackle growing complexity. High-level synthesis tools that turn a high-level algorithm description into low-level RTL

code are gaining popularity in academia and industry as a way to increase the degree of abstraction in hardware designs. High-level synthesis tools can be classified into three categories that are based on hardware-description language, C-like language , and high-level language.

Hardware-description language. Bluespec [84] is a hardware-description language that allows designers to specify hardware with the same accuracy as RTL but in a more compact form. Similar to Verilog, Bluespeculies requires the definition of modules, wires, and blocking and non-blocking assignments for specifying hardware components. Bluespec provides more expressive types, overloading, encapsulation, and flexible parametrization to facilitate the reuse of code. Using the Bluespec compiler, RTL descriptions corresponding to Bluespec programs are generated.

Designers may decide to use languages developed for a specific application domain like C, C++, SystemC, and OpenCL. These languages benefit from a compact and high-level syntax, which is very efficient for representing the target code. They are used in almost all high-level synthesis tools from CAD vendors (Cadence C-to-Slicing Compiler, Synopsys Synphony C Compiler, Mentor Graphics Catapult C, Intel Quartus, and Xilinx Vivado). The quality of the RTL designs created by synthesis processes is highly dependent on the quality of the input source code.

C-like language. Designers may decide to use languages developed for a specific application domain like C, C++, SystemC, and OpenCL. These languages benefit from a compact and high-level syntax, which is very efficient for representing the target code. They are used in almost all high-level synthesis tools from CAD vendors (Cadence C-to-Slicing Compiler, Synopsys Synphony C Compiler, Mentor Graphics Catapult C, Intel Quartus, and Xilinx Vivado). The quality of the RTL designs created by synthesis processes is highly dependent on the quality of the input C-like programming language.

High-level language. There are languages that allow designers to retain control over all the details of the generated hardware. These languages leverage more complex constructs and more powerful semantics to enable a compact and less error-prone description of the target design. Chisel [6] is a popular example of these languages, which define the necessary data types and constructs to embed hardware descriptions in Scala. Chisel provides Scala libraries for defining hardware datatypes as well as procedures for compiling source code into either a cycle-accurate C++ emulator or a Verilog implementation.

1.3.3 Design space exploration

Design space exploration (DSE) is the process of determining the design solutions that best meet the required design criteria. DSE methods should come up with designs quickly while meeting the given requirements (e.g., power, performance, and area). This kind of exploration is obviously complicated, as the search may involve tentative designs that come from choosing specific parameter values, configurations, or even algorithmic alternatives. Most of the time, developers can evaluate a design in two ways: empirically (execution/simulation) or analytically (modeling). Designs can be either single-objective or multi-objective with respect to the optimization criteria. A single-objective optimization problem aims to identify the optimal solution for a single criteria or metric, such as execution time, or a combination of this metric with energy consumption or power dissipation metrics. A multi-objective optimization problem is one where you have to find the best solutions for many different, often competing, goals.

In some cases, an exhaustive search can be conducted, or heuristic methods can be used to find the best design solutions. However, in the vast majority of cases, an exhaustive exploration of this space, even if it is done automatically, is not possible. Different approaches to DSE use different kinds of algorithms, including random search, evolution, and swarm algorithms, as well as machine learning [16]. Due to the complexity and speed of this search field, automation of design choice selection, creation, and evaluation is essential. Existing work for DSE utilizing CAD tools can be divided into four approaches:

— First of all, local-search heuristics are the most common since they can be used to find the Pareto-optimal ones [5]. These techniques generate a high number of trial synthesis tasks in each iteration, but a major fraction of the synthesis outputs are discarded as the algorithm progresses.

— The second strategy predicts the quality of results (QoR) rather than invoking actual synthesis jobs. The iterative-refinement framework outperforms local-search-based frameworks in general [78]. By merging accelerator-core design with commercial HLS tools and industrial physical-synthesis tools, the effectiveness of DSE is confirmed.

— The third approach entails pre-pruning the design space to reduce the search space. This approach still requires significant design skill and is limited to a small set of applications [102].

— The fourth approach derives Pareto points from well-defined "knob-setting to QoR" mapping functions [108].

Electronic design automation (EDA) tools are important to the evolution of the Very Large-scale Integration (VLSI) industry. Currently, as the technological node scales down, design complexity continues to rise. In order to ensure timing closure, reliability, and manufacturability, more sophisticated algorithms and optimizations have been integrated into EDA tools, therefore increasing their complexity. The purpose of CAD tools is to determine the settings in CAD tool scripts that, after synthesis, result in optimal circuit performance. It can be stated intuitively as an optimization issue, but the synthesis process is too complex to be modeled analytically. The solution space cannot be fully searched by designers, nor can they deduce a closed-form solution. In the design space, optimization should be addressed in an exploratory manner.

Bayesian hyperparameter optimization

The field of architecture has found many uses for machine learning (ML), and it is now widely used for tasks such as design, optimization, and simulation [121]. Numerous components, such as the core, cache, NoC, and memory, have already achieved success with ML, with performance frequently exceeding that of previous state-of-the-art analytical, heuristic, and human-expert solutions. These approaches are more powerful than heuristic optimization in terms of convergence and the quality of the solutions that are obtained. Bayesian hyperparameter optimization (also known as sequential model-based optimization, SMBO) should be considered carefully when applying machine learning to design space exploration problems. The model is trained using input features and output targets, with the result being a model that can predict the output of new, unseen inputs. The strategy is to construct a probability model of the objective function, which is then utilized to choose the most promising hyperparameters for evaluating the true objective function. Typically, the case for computer system workloads consists of several discrete variables, i.e., categorical (e.g., boolean) or ordinal (e.g., choice of cache sizes), over which derivatives cannot even be formed. This approach is referred to as black-box optimization and design space exploration in the computer systems world.

Machine learning has evolved into an effective tool in the field of computer architecture. A survey by Penny et al. [91] shows that it can be used for design, optimiza-

tion, simulation, and many other design processes. These techniques offer interesting opportunities for architecture simulation, especially in the early stages of the design process. As an example, Bhardwaj et al. present in [12] a Bayesian optimization-based framework for determining the optimal hybrid coherency interface for many-accelerator SoCs in terms of performance. Due to trade-offs between different criteria, most computer architecture designs need multi-objective optimization (power, performance, area, etc.). These are some of the many factors that have led to a new trend: multi-objective hardware design, in which a single architecture is optimized for multiple metrics simultaneously. HyperMapper [82] handles multi-objective optimization, unknown feasibility constraints, and categorical and ordinal variables. This new approach uses the user's past knowledge when it is available. These features are common in computer systems but not so common in systems made for space exploration. The suggested method uses a white-box model, which, unlike neural networks, is easy to understand and can help people understand the result produced by an automatic search. They use and study the new technique to automate the static tuning of hardware accelerators in the recently announced Spatial programming language, with the goal of minimizing design run-time and computing logic while fitting on a target FPGA chip. HyperMapper gives better Pareto fronts than state-of-the-art baselines, better or comparable hypervolume indicators, and an 8x improvement in sampling budget for most benchmarks.

1.4 Summary

The crisis caused by the scaling of technologies has pushed the semiconductor industry in the direction of adopting innovative and emerging technologies. Because of this shift in paradigm, the idea of a system-on-chip, or SoC, came to be characterized as a design for a single chip that integrates numerous different kinds of components. Heterogeneity makes SoC design and programming more difficult. For a few skilled engineers who understand system-level trade-offs, how to build appropriate hardware, and how to handle memory and power management, this may be a challenging job. So, designers need tools that make it easy for them to develop heterogeneous SoCs.

Simulation is often used in the design of new computer architectures or in the study of existing ones. It was found that an FPGA-accelerated simulator can improve the speed of simulations and even make them possible to run on a powerful server using

FPGAs. FPGA-accelerated simulators have great potential for heterogeneous system design. In hardware design and computer architecture, the issue of looking for complex design spaces while considering simulation as a black-box function arises. As the complexity of the SoCs develops, so does the importance and difficulty of finding the optimal design. We explore the use of Bayesian optimization to solve a hardware design challenge.

Our research aims to contribute to the design process by helping with the development of new SoC designs that combine more heterogeneous components. The result is a flexible design methodology that promotes simulator reuse and supports system design space exploration. The following three chapters will highlight three major contributions I made during my PhD: FPGA-accelerated simulation; integration of a pre-RTL accelerator model into the FPGA-based simulator; and determining the optimal configuration architecture for heterogeneous SoCs.

CHAPTER 2

FPGA-ACCELERATED SIMULATION OF HETEROGENEOUS ARCHITECTURES

This chapter mainly focuses on FPGA-based simulation, specifically building experiments to explore the capabilities of the HAsim FPGA-accelerated simulator. First of all, an overview of the HAsim framework is presented. We introduce LEAP, an operating system that provides a platform for developing FPGA-based applications, and the HAsim simulator. We also present Bluespec System Verilog and its simulator, which is used to specify and verify simulation models on HAsim . Secondly, we present a case study on the building of RISC-V processor models (unpipelined, in-order-pipelined, and out-of-order). Several simple benchmarks have been defined to make sure that our models of processors are correct. Thirdly, we exploit the fact that the simulator is independent of the hardware platform by running experiments on AMD/Xilinx and Intel/Altera FPGAs. In an effort to deploy the FPGA-based simulator on an Intel CPU-FPGA platform, we implemented a communication channel between the CPU and the FPGA. The communication channel has been evaluated through a series of experiments. Finally, we illustrate the FPGA-accelerated microarchitecture simulation challenges by describing the obstacles to approaching and developing them.

Contents

2.1 Introduction **38**
2.2 FPGA-based processor simulation with HAsim **40**
2.2.1 HAsim framework overview 42
2.2.2 The LEAP operating system for FPGA-based applications . . . 43
2.2.3 Bluespec system verilog 46
2.3 A case study with the design of RISC-V models within the HAsim framework **49**
2.3.1 Semantic of function partition 50
2.3.2 Timing model creation 51
2.3.3 Evaluation results 54
2.3.4 Targeting the Xilinx Virtex-7 FPGA platform 56
2.4 Deploying the HAsim simulator on a Intel CPU-FPGA platform . . **58**
2.4.1 Intel Xeon+FPGA platforms 59
2.4.2 Implementing communication channels support for Intel CPU-FPGA platform 60
2.4.3 Validation 63
2.5 FPGA-Accelerated microarchitecture simulation challenges . . . **65**
2.6 Conclusions **70**

2.1 Introduction

When considering designing heterogeneous architectures, the number of possible design combinations leads to a huge design space with subtle trade-offs and design interactions. To determine which design is optimal for a given application, it is necessary to simulate with accuracy many possible solutions. Although software-based simulation has significantly contributed to the validation of computer architecture research, its capacity to rapidly evaluate design space points may be reducing. Tan et al. [113] argue the move to multi-core has hampered the performance of pure software-based simulators since multi-core simulation targets exhibit complex timing-dependent non-deterministic behavior. Such behavior needs detailed cycle-level simulation. Unfortunately, software-based simulators are notoriously challenging to parallelize, that is to say, to take advantage of multi-core host machines, and as a result, their performance continues to decline.

To tackle this simulation gap and perform cycle-accurate simulation quickly, perhaps the only way is to use dedicated hardware to accelerate the most complicated parts of the target architecture model. Several research groups have been exploring the use of hardware to build various forms of hardware-accelerated architecture simulators. There are successful projects that use FPGA to implement all or part of the simulation (usually micro-architecture and interconnect models at least), and the industry makes extensive use of large FPGA-based modeling and prototyping systems [111]. This research has led to FPGA accelerated model execution (FAME) techniques, where the desired target architecture is mapped to an FPGA for evaluation [113]. Another example of specialized hardware is the use of Graphics Processing Units [95]. Although FPGA can help improve simulators' performance, designing a performance model targeting an FPGA is more complicated than designing a software simulator. Moreover, once the FPGA has been designed, it also needs to be debugged. Therefore, the risk is that the idea of the deployment of performance models on FPGA will fail not because of a lack of performance but because of increased model development time. To overcome these development challenges, we advocate for a platform that offers a set of module abstractions that simplify the construction of FPGA-based simulators within a convenient infrastructure.

latency-insensitive environment for application programming (LEAP) [45] was designed at Massachusetts Institute of Technology in 2014. It offers fundamental device

abstractions for FPGAs and a suite of standard I/O and memory management utilities to address these concerns. LEAP works as a software-based operating system by providing standard, abstract interfaces to underlying hardware resources, automated resource management, and efficient system libraries. Thanks to a powerful compiler that supports automated implementation decisions, LEAP can be very efficient. LEAP is compatible with several FPGA platforms from AMD/Xilinx and Intel/Altera. It was exploited by Intel and MIT for a variety of FPGA projects, including the HAsim performance modeling framework [87], an H.264 decoder [46], an OFDM framework [83], an SSD [67], and other projects.

The classical design methodology of a software-based simulator generally consists of a partition into a functional model that is responsible for correct ISA-level execution of the computer system and a timing model that predicts performance and other metrics. In the context of FPGA-based simulators, partitioning along the functional and timing boundaries allows for a variety of mapping choices [79, 24]. By selecting which parts of the simulator are mapped to the FPGA and how these components interact, different simulator organizations can be envisioned. HAsim is based on the principle of a timing-directed simulation [88]. It places both partitions on the FPGA in order to minimize communication latency. It uses the bluespec system verilog (BSV) [84] high-level synthesis language and runs on top of the LEAP virtual platform infrastructure, which allows it to run on different FPGAs without recoding. HAsim is designed to construct efficient simulators from a library of reusable components rather than emphasize any particular target processor. The key idea is to reduce development efforts by reducing the amount of code the architects must change to perform their design space exploration.

Besides, cloud service providers are investing more in FPGA infrastructure to offer it to clients. Amazon web services (AWS) [1] and Alibaba Cloud [2], both using AMD/Xilinx FPGAs, started offering FPGA-based infrastructure as a IaaS (Infrastructure as a Service). Likewise, Intel is developing a family of FPGA accelerators for data center applications. Through the Intel hardware accelerator research program (HARP) [58] and the IL academic compute environment (IL ACE) [3], they are making a variety of platforms available to academics. In 2016, Intel initiated the HARP program, consisting of a single chip combining an FPGA and an Intel Xeon CPU. It is designed to accelerate

1. https://aws.amazon.com/ec2/?nc1=h$_{ls}$
2. https://eu.alibabacloud.com/en
3. https://wiki.intel-research.net/Introduction.html

the development of new technologies. It enables a complete redesign of the traditional architectures used for accelerated computing to be accomplished. The IL academic compute environment (IL ACE) enables and advances academic research in a wide range of fields. It is now comprised of servers powered by Intel Xeon processors and FPGAs. The Intel platform makes it possible to create an innovative hardware solution that properly adapts to the software that is being run. The open programmable acceleration engine (OPAE) [4], as well as a common hardware-side core cache interface (CCI-P) [5], are shared by the whole family of processors. This design is intended for large workloads in data centers as well as application-specific accelerators. According to researchers at Stanford University's Graduate School of Computer Science, cloud service providers are investing more in FPGA infrastructure, which will enable access to more powerful platforms. An objective of the research is to determine whether cloud-based FPGAs can accelerate computer architecture simulations. The following are the contributions presented in this chapter:

— Through a case study, we demonstrate the potential of the HAsim framework for the building of performance models for the RISC-V instruction set defined at Berkeley. By using the HAsim framework, we modeled an unpipelined, in-order-pipelined, and out-of-order RISC-V processor and evaluated their performance.

— The study of FPGA-based application support platforms is independent of the physical platform. A cloud-based simulator has been developed using the Intel CPU-FPGA. Using OPAE and the CCI-P software libraries, we have developed a communication channel that enables FIFO-governed bidirectional data exchange between the Xeon processor and the FPGA.

2.2 FPGA-based processor simulation with HAsim

The research accelerator for multiple processors (RAMP) project was started in 2005, and it aimed to create a shared full-system simulation infrastructure for studying thread-parallel machines. In section 1.2.4 of Chapter 1, four FPGA-based simulators are described. These are the most popular RAMP projects and are summarized in Table 2.1. ProtoFlex is a simulator at the architecture level with 16-way host multithreading

4. https://github.com/OPAE
5. https://www.intel.com/content/www/us/en/docs/programmable/683190/1-3-1/core-cache-interface-cci-p.html

of a single functional model hosted on an FPGA. Through a process called "transplantation," ProtoFlex could switch between running on an FPGA and running on a CPU. FAST is a cycle accurate x86 simulator that uses a functional model hosted by the CPU and a timing model hosted by the FPGA. RAMP-Gold uses 64-way host multithreading and timing and functional models hosted on an FPGA.

	FAST-2007[22]	**Protoflex-2009**[26]	**RAMP Gold-2010**[113]	**HAsim-2011** [87]
Open Source	No	Yes	Yes	Yes
Cores/Speed	1/ 1 MIPS	16/ 62 MIPS	64/ 50 MIPS	16/ 1.6 MIPS
Cycle Accurate	Yes	Yes	Yes	Yes
Network on Chip	No	No	Yes	Yes
Code Reuse	No	No	No	Yes
Full System Simulation	Yes	Yes	Yes	Yes
Multi-threading	No	Yes	Yes	Yes
Programming Language	Bluespec	Bluespec	HDL	Bluespec
Functional/Timing	CPU/FPGA	FPGA/CPU	FPGA/FPGA	FPGA/FPGA
Simulator Architecture	SFF	Functional	Timing-directed	Timing-directed
Comments	Bottleneck	Bottleneck	No Pipeline & NoC	Trade-off accuracy & speed

Table 2.1 – Comparison of Research Accelerator for Multiple Processors (RAMP) projects.

Likewise, HASim also used timing and functional models that were implemented on an FPGA. Its pipeline and memory hierarchy models were more detailed. In addition, research was conducted to determine how to divide HASim across multiple FPGAs. Using two FPGAs allowed virtual instances to share resources more efficiently, allowing HASim to host eight times as many cores. HAsim appears to be a good choice for a depth study for the following reasons:

— Both partitions (functional and timing) are placed on the FPGA. This leads to a more traditional partitioning, whereby the functional partition does not need to speculate as to the timing model's direction as it can receive feedback directly after every instruction.

— HAsim uses a migration scheme so that rare events that are difficult to place on the FPGA, such as system call instructions, can be farmed out to software regardless of whether they occur in the functional or timing partition. However, this requires being able to interface with a software simulator.

— Compared to RAMP Gold, HAsim has many similar characteristics (time-directed architecture, functional/timing on an FPGA, time-multiplex simulator, etc.). However, RAMP Gold does not model a realistic core pipeline and does not model micro-architectures that use branch prediction or out-of-order execution.

— HAsim also approaches reducing development effort. It enables the reuse of source codes, uses a high-level hardware description language, and is built on top of a virtual platform, which makes FPGA easier to use.

2.2.1 HAsim framework overview

The "Hardware-based Architecture Simulator" (HAsim) is an open-source infrastructure developed at MIT, mainly written in Bluespec System Verilog. It is used to build processor models and has a suitable environment for creating timing models for pre-RTL FPGAs. The structure distinguishes between functional and timing components. Timing is separated from the FPGA clock, which enables hybrid computing, in which software handles complex but infrequent tasks. HAsim is built on top of a virtual platform and is used to provide operating system services to FPGAs. This virtual platform is LEAP, and it can be used for a wide range of applications.

HAsim is designed to construct efficient simulators from a library of reusable components rather than emphasize any particular target processor. The key idea is to reduce development efforts by minimizing the amount of code the architects must change to construct their space exploration designs. HAsim is divided into four major components, as shown in Figure 2.1:

— The functional partition handles correct ISA level execution of the instruction stream.

— The timing partition (or timing model) is responsible for tracking micro-architectural specific timings (such as branch predictors and cache misses).

— A collection of predefined modeling components that serve as a library (such as branch predictors and caches).

— The unmodeled component refers to all functionality not directly related to simulation (including the ability to track statistics and parameters, as well as the virtual platform necessary to interact with the host CPU).

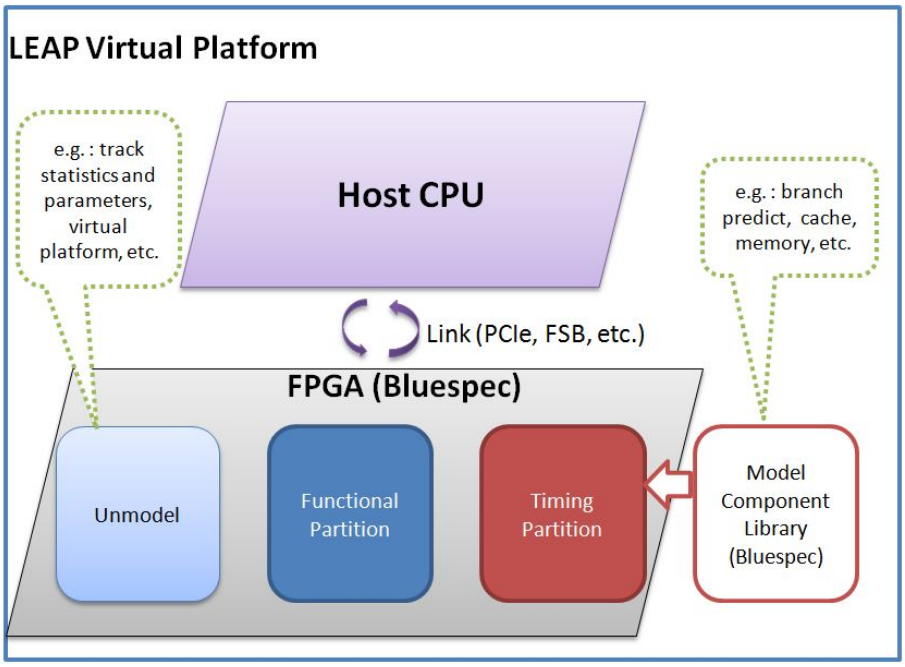


Figure 2.1 – HAsim Framework Overview.

HAsim is a hybrid model comprising code running on a general-purpose host CPU as well as on an FPGA. This scheme leverages the strengths of each physical platform: the FPGA for fine-grain parallelism and the CPU for rare but challenging implementation events, such as system calls.

2.2.2 The LEAP operating system for FPGA-based applications

FPGAs provide significant power and performance advantages over more traditional sequential architectures for a wide range of applications. In spite of these benefits, FPGAs have only been used in a small number of specialized fields so far. FPGA programmers are currently exposed to all of the essential system details that software operating systems have long since abstracted away. The problem is that this makes it impossible for FPGAs to be used in more general systems because of the difficulty of programming them. Latency-insensitive Environment for Application Programming (LEAP) [6] is a set of modules that provide a straightforward foundation for developing FPGA-based applications. It is similar to an operating system, but its compilation sup-

6. https://github.com/LEAP-Core

port is more extensive. Interfacing with the physical devices attached to the FPGA is one of the most challenging aspects of FPGA development. LEAP abstracts physical devices into a number of fundamental abstraction layers. On top of these abstraction layers, LEAP offers a library of generally helpful services. Because the LEAP virtual platform offers a collection of virtualized device abstractions, FPGA developers may focus on implementing core functionality rather than debugging low-level device drivers. LEAP provides a library of generally useful services built on top of these abstraction layers. LEAP facilities include communication services, memory services, and latency-insensitive channels.

Communication services

When instantiating modules, most hardware-design languages impose a rigid communication hierarchy. This has the unintended consequence of adding charges to a child's interface and leading to changes for the parents. Pellauer et al. [86] solve this problem by "softening" the rigid communication structure, hence the name "soft connections" for their technique. The user can define connection endpoints that are automatically connected at compilation time rather than by the user. This technique preserves modularity by allowing the designer to specify a logical communication topology that is distinct from the physical implementation. The user does not link the endpoints; instead, static elaboration is used to do it automatically. Soft Connections maintains modularity by allowing individual modules to be swapped out without affecting the instantiation hierarchy as a whole. They are made up of three basic primitives: send, receive, and chain, each of which can be customized in a variety of ways. Send-Receive pairs behave similarly to FIFO channels, whereas chains are broadcast primitives.

To facilitate typed communication methods between an FPGA and an external software process, LEAP offers a typed asynchronous request-response protocol known as Remote Request Response (RRR). The user creates services whose servers exist on either the FPGA or in software, with the client located on the other end of the communication chain. The interface that each server exposes is defined by the end user. At compile time, RRR stub compilers create the marshaling, demarshalling, and multiplexing code that is used to connect the user code to the underlying LEAP communication channels, which is then executed by the user code. The RRR interface abstracts away almost all of the complexities related to communicating between an FPGA module and a software module, allowing for more efficient communication.

Memory services

LEAP scratchpads are hierarchical storage structures that provide the same interface as block RAMs but allow the allocation of larger arrays than what an FPGA can support. They are a form of abstraction that creates and maintains numerous self-contained memory arrays inside a massive underlying store in a dynamic manner. Automatic caching of scratchpad accesses happens at many levels, from set-associative caches based on shared on-board RAM to private caches stored in FPGA RAM blocks. In the LEAP framework, scratchpads are plug-in replacements for on-die RAM blocks. Additional libraries facilitate the management of heaps inside a storage set. Similar to how software is made, LEAP scratchpads let the architect of hardware focus more on basic algorithms and less on managing memory. HAsim makes use of LEAP scratchpads to facilitate the simulator's scaling to larger multicore targets. Usually, the data space needed to simulate the CPU caches is too large to fit on the FPGA.

Latency-insensitive channels

LEAP separates logical and physical communication using latency-insensitive channels (LI) to make communication between the FPGA and CPU easy and portable. These channels are implemented as simple FIFOs in hardware. Designs are built such that processing only occurs when data is available in input FIFOs or when output FIFOs have sufficient space. As with concurrent FIFO modules, latency-insensitive channels provide basic enqueue and dequeue operations as well as status methods (such as *notFull* and *notEmpty*) for the user application to consider in deciding when to transmit and receive data.

The structure of the virtual platform used by HAsim is shown in 2.2. It illustrates the fundamental interactions between the simulator and the platform. Multiple distributed services on the CPU and the FPGA are able to communicate with one another thanks to a collection of virtual devices and a communication protocol called Remote Request-Response (RRR). The key advantage of this method is its mobility. Without modifying the program, the virtual platform may be transferred to a new physical FPGA platform. Developers just have to rewrite low-level device drivers. LEAP provides a common set of interfaces for the FPGA to communicate with the outside world, which reduces development time. Thus, it is expected that the majority of FPGA-based simulators will be hosted on hybrid computing platforms consisting of one or more FPGAs and one or

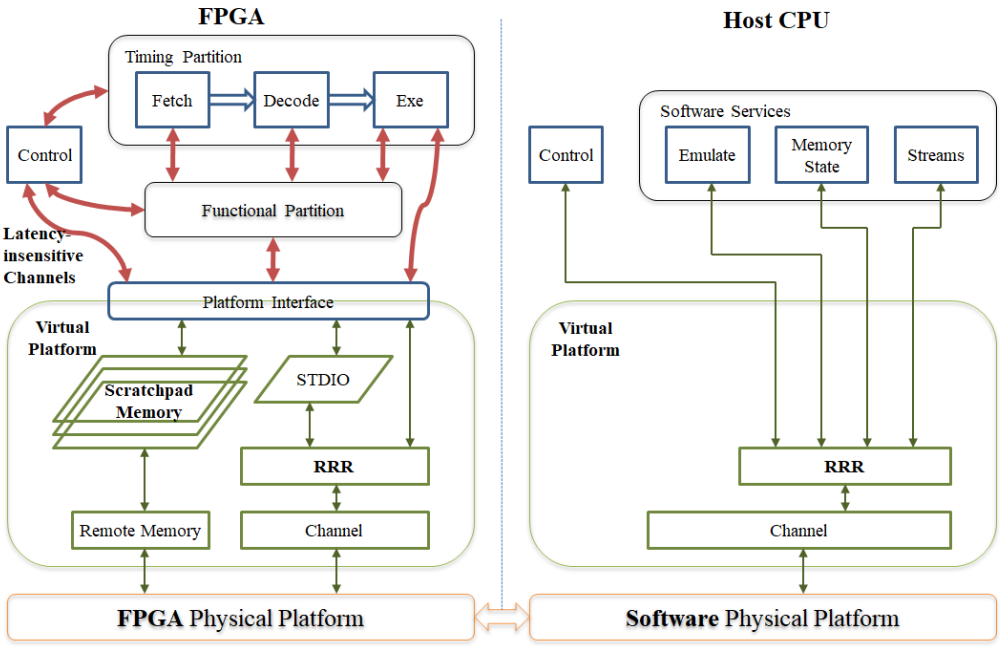


Figure 2.2 – The HAsim simulator is based on the LEAP Virtual Platform [65].

more CPUs. In a hybrid CPU/FPGA system, the platform is becoming more accessible for sharing tasks between the FPGA and the CPU through communication protocols such as remote procedure calls (RPC) and shared memories.

2.2.3 Bluespec system verilog

Bluespec SystemVerilog (BSV) is among the first efforts to develop a higher-level hardware description language than Verilog/VHDL. Bluespec is based on System Verilog syntax and allows a high degree of abstraction in both behavioral and structural descriptions. Defining modules, wires, and blocking and non-blocking assignments in Bluespec is similar to Verilog. The movement of data from one state to another is described by *rule*. Rules consist of two components: *rule* conditions, which are boolean expressions that determine when the *rule* is enabled, and *rule* bodies, which are actions that describe state transitions. Bluespec substitutes processes with atomic *rule*, from which control logic is automatically inferred as a step toward HLS. For example, assume that two rules give the same wire a value based on different conditions. If both conditions are met in the same clock cycle, the compiler generates the logic to avoid the

conflict, which would otherwise result in a double-driven signal. The Bluespec compiler can perform a partial schedule of the code to determine which *rule* run concurrently and which do not. However, the designer is still responsible for the specification and timing of a *rule's* behavior.

The core of BSV is formed of modules and interfaces. Modules and interfaces make up the actual hardware. An interface contains members known as methods. To begin, a method is similar to a function in that it is a procedure that takes zero or more arguments and returns a result. As a result, method declarations within interface declarations mimic function prototypes. Each method becomes a bundle of wires when translated into RTL. Module instances are arranged in a hierarchy. A module definition may include instantiation specifications for other modules and their interfaces. Within a module, a single module definition can be instantiated multiple times. Let us illustrate the foundation of BSV with the example of a counter, as shown in Figure 2.3.

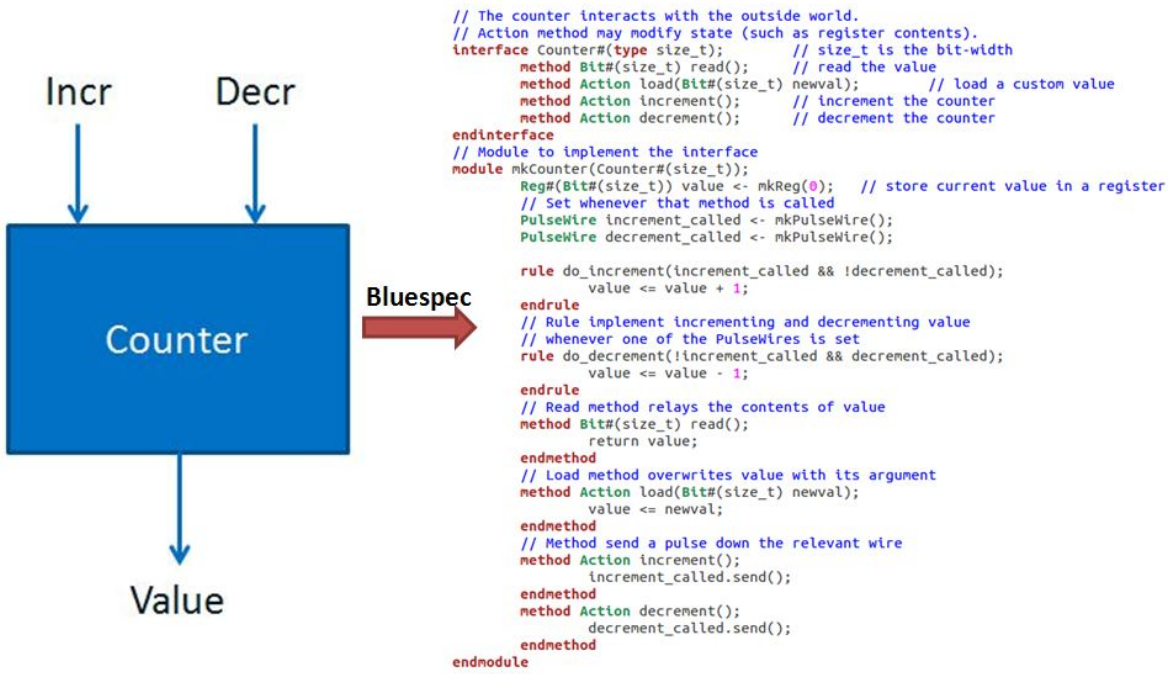


Figure 2.3 – A Counter expressed in Bluespec.

The counter is a register (the state). The increment *Incr* and the decrement *Decr* are operations on the register. Bluespec provides a way to declare a state, define operations on the state, wrap the state/operations in modularity, and define operations on

modules. First, an interface is defined, and it has two different kinds of methods: one that returns a value *read()*, and two that perform actions: *increment()*, *decrement()* and *load()*, respectively. The module implementing the Counter interface is then defined. This module has the name *mkCounter()*. In Bluespec, *rules* are used to describe how data is moved from state to state. It may fire when the predicate is true, the entire rule executes in one cycle, and side effects are visible at the start of the next cycle. The method is one of a module's public functions and is invoked by the parent module's rules or other methods. By combining the register instance with the methods and placing them all inside of the *module* and *endmodule* keywords, we are able to obtain the full counter module. Running this program will allow us to convert our BSV into Verilog.

The design flow that is used in this work, illustrated in Figure 2.4 uses Bluespec System Verilog (BSV) and its compiler to provide rapid simulation of a hardware system and its FPGA implementation. The BSV compiler generates efficient RTL code that manages all possible interactions between rules by inserting the necessary arbitration and scheduling logic, which would otherwise need to be manually designed and coded. Using BSV syntax, the compiler produces a hardware description in either Verilog or Bluesim.

The entire architectural design is implemented using BSV, and the simulation results were analyzed using Bluesim, a cycle-accurate BSV simulator, to investigate the system's correctness. Bluesim has been utilized for accurate analysis of simulation results generated in value change dump (VCD) format by the Bluespec compiler. Using BSV synthesis, Verilog files corresponding to each implemented BSV module are then generated. The bit stream file is then generated using synthesis tools (Vivado, Quartus, etc.) to generate Verilog files. The output bit stream file was then loaded into the FPGA, and the entire system's functionality was evaluated in real time.

The Bluesim simulator converts Bluespec designs into C++ objects that implement the rules and methods defined in each module and store any relevant data and temporary variables. In addition to the modules, scheduling routines are also generated to coordinate the application of rules across the whole design. The compiler includes a library that implements fundamental program components like modules, functions, and system tasks. Bluesim involves C++ header files (.h), C++ source code files (.cxx), compiled object files (.o), and compiled shared object files (.so). When the executable files are called, the default service begins timing the circuit and running the program. Bluesim was used to analyze the performance of the three processor models presented

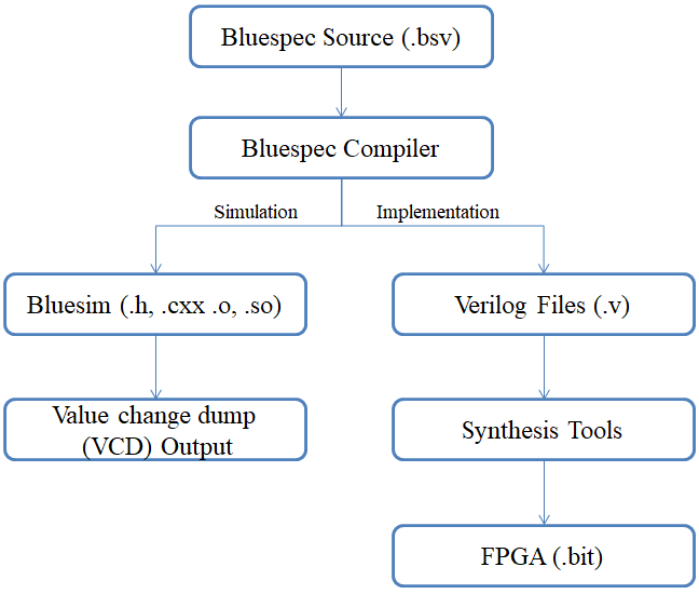


Figure 2.4 – Design flows target FPGA platforms using Bluespec System Verilog.

in section 2.3.2, which were developed in BSV. The Ventor synthesis tools (Vivado and Quartus) have been employed for the purpose of producing the bit stream files for FPGAs as well as the synthesis of the Verilog files that have been generated by the Bluespec compiler. Our study utilized the Xilinx Virtex-7 and Intel BDX platforms as FPGA target platforms.

2.3 A case study with the design of RISC-V models within the HAsim framework

RISC-V ("risk-five") [119] was originally developed at the University of California, Berkeley, in 2010. RISC-V is an instruction set architecture (ISA) that was originally designed to support computer architecture research and education, for which it could become a standard open architecture for industry implementations. Instead of being an open-source CPU, RISC-V is an ISA specification. Unlike the RISC instruction set architectures on which it is based, RISC-V is freely available for use by any party via a permissive free software license and a patent grant. They help to make sure that software can be re-used across several chip designs and to develop open standards

for whole platforms, including I/O and accelerators. RISC-V claims that the ISA is ideal for a wide range of uses, including embedded systems, low power applications, and industrial uses. Due to these advantages, it is a suitable target for the case study.

Through a case study, we show how the HAsim framework could be used to build performance models for the RISC-V [7] instruction set, which was released to the open source community. The traditional way to make a software-based simulator is to decompose it into a functional model that makes sure the computer system runs correctly at the ISA level and a timing model that predicts performance and other metrics. We present in this section the implementation of RISC-V processor models and show how code reuse facilitates the design of two timing models for different target processors that can use the same functional partition and assess their performance.

2.3.1 Semantic of function partition

The role of the functional partition is to calculate the new state of the processor after executing an instruction. It consists of eight operations, as shown in Table 2.2, corresponding to traditional microprocessor pipeline stages. The timing partition invokes the operations and determines the state of the machine in the order. The operations are used by the timing model to produce a cycle accurate simulation. The same functional partition can be reused across different timing models to simulate different microarchitectures. The operations are typically invoked in the order specified for a single instruction. This corresponds to instructions flowing through pipeline stages in a real computer: the instruction is fetched *(getInstruction)* before it is decoded *(getDependencies)*, which takes place before register reading *(getOperands)* and so on. The order in which the timing model invokes these operations on instructions determines the state of the machine.

Most of the components needed for the specification of the functional partition come from existing HAsim modeling projects at MIT. They can be easily instantiated by way of the AWB tool [41]. In order to design RISC-V models, only information related to instruction decoding (number of sources, destinations, and barrier information) and the hardware datapath for executing common instructions has to be specified. This is the advantage of the ISA-independent datapath feature of HAsim.

7. https://riscv.org/

Table 2.2 – Functional partition operations.

Operation	Behavior
getInstruction	Get memory address, return corresponding instruction.
getDependencies	Allocate the destination physical register, look up physical registers containing the operands.
getOperands	Read the physical register file and the instruction, return opcodes and immediate operands.
getResult	Execute the instruction, return the result including branch information, effective address.
doLoads	Read the value from memory, write to register.
doStores	Read the register file, write the value to memory.
commit	Commit the instruction's changes, remove instruction.

2.3.2 Timing model creation

Three target architectures are considered in this work, which are defined to be used with the RISC-V functional partition: an unpipelined processor, an in-order pipelined processor, and an out-of-order processor. Three timing models operate the same operation, as shown in Figure 2.5. Each model does the same amount of fundamental work, and the only change is related to the time modeling. Every operation in the timing model of an unpipelined processor always takes one cycle to run. It needs 15 model cycles to execute. Following that, we have a simple in-order pipeline timing model. This model stalls between instructions *0x101* and *0x102* because of a hazard called "read after write." Assuming a perfect memory hierarchy and a one-cycle ALU, it takes 9 model cycles to execute this sequence of operations. An out-of-order, 2-way superscalar model executes multiple operations on the functional partition before advancing the model clock cycle. If a branch is stalled on a dependency, the out-of-order model predicts that the branch will not be taken and will issue past the stalled branch..

The advantage of the timing model is being able to reuse the same functional partition across multiple timing models to simulate multiple microarchitectures. The timing model does not need to implement all structures, as the functional partition handles the functionality. Figure 2.6A shows the specification to be used in a basic processor timing model for a processor that doesn't use pipelines (an unpipelined processor) and runs each instruction in a single clock cycle. Each functional partition action is carried out by the implementation before the model time counter is incremented. The actions that

0x100 ADDI r1 := r1+1 0x101 ADD r2 := r3+r4 0x102 SUBI r2 := r2-1			
Cycle	**Unpipelined**	**In-order pipelined**	**Out-of-Order**
0	getInst(0x100)	getInst(0x100)	getInst(0x100) getInst(0x101)
1	getDeps(0x100)	getInst(0x101) getDeps(0x100)	getInst(0x102) getDeps(0x100) getDeps(0x101)
2	getOps(0x100)	getInst(0x102) getDeps(0x101) getOps(0x100)	getDeps(0x102) getOps(0x100) getOps(0x101)
3	getResult(0x100)	getDeps(0x102) getOps(0x101) getResult(0x100)	getOps(0x102) getResult(0x100) getResult(0x101)
4	commit(0x100)	getResult(0x101) commit(0x100)	getResult(0x102) commit(0x100) commit(0x101)
5	getInst(0x101)	commit(0x101)	commit(0x102
6	getDeps(0x101)	getOps(0x102)	
7	getOps(0x101)	getResult(0x102)	
8	getResult(0x101)	commit(0x102)	
9	commit(0x101)		
10	getInst(0x102)		
11	getDeps(0x102)		
12	getOps(0x102)		
13	getResult(0x102)		
14	commit(0x102)		

Figure 2.5 – An example of three different timing models operating on the same instruction set.

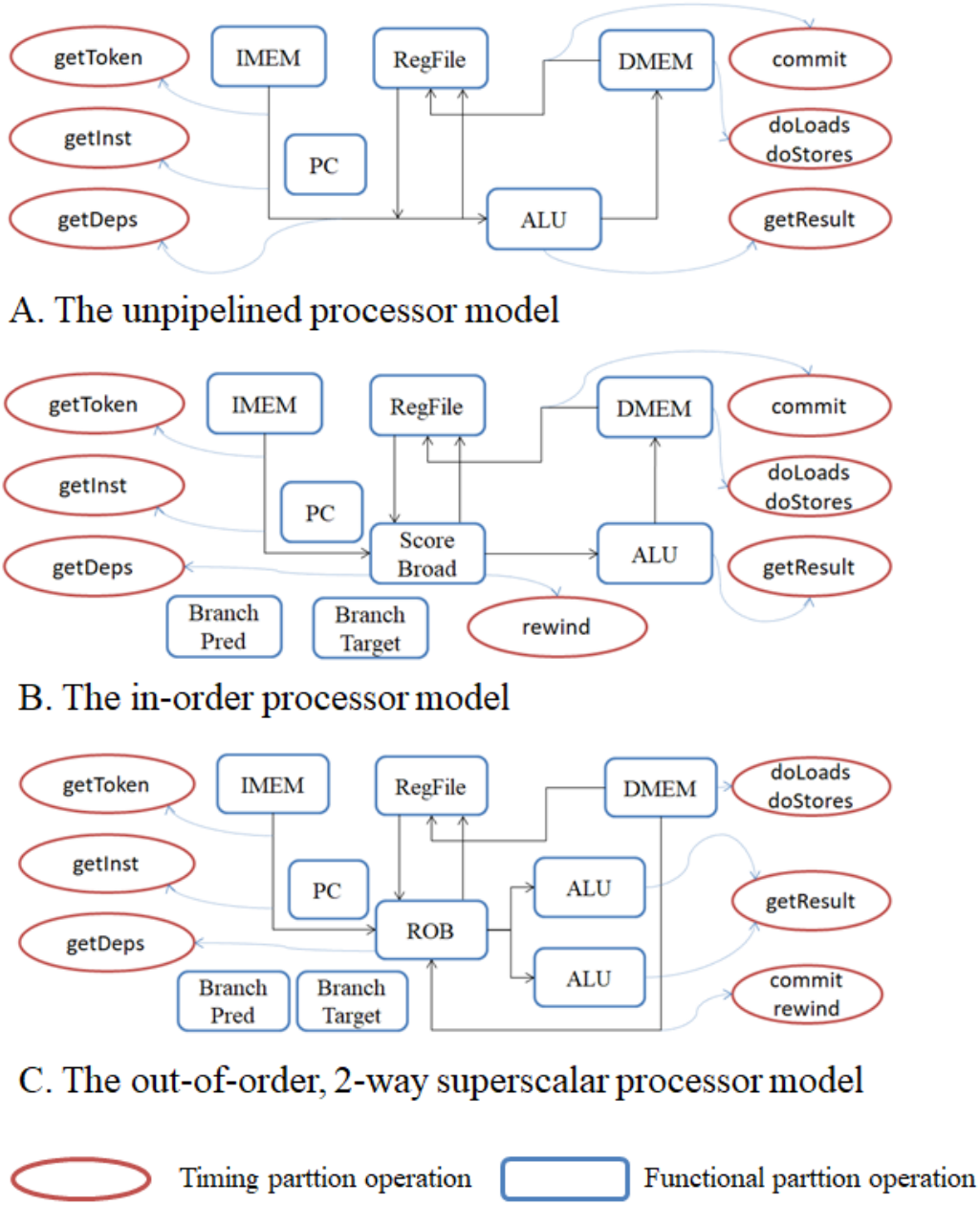


Figure 2.6 – Target processors and their simulator implementation.

make up the timing model are referred to as getToken(), getInst(), getDeps(), getResult(), doLoads(), doStores(), and commit(). All of the simulation work for the different components of the processor (ALU, IMEM, and DMEM) is done via functional partitioning operations. Figure 2.6B shows an in-order processor target. The branch predictor structure is implemented entirely in the timing model, as it controls which address the timing model will pass to getInstruction(). On mispredictions, a rewind() is issued to represent a pipeline flush.

As shown in Figure 2.6C, an out-of-order, 2-way superscalar processor is built as an extended version of the unpipelined timing model. The functional partition is called before launching model time to simulate superscalar behavior. In particular, the getResult() operation is called four times, and the ALUs do not need to be used. The simulated Reorder Buffer (ROB) is considerably simpler than a real ROB, as it does not implement dependency tracking logic. Instead, it uses the result of the getDeps() operation and then uses a sequential search to determine which instructions should be issued next.

2.3.3 Evaluation results

The three processor models (unpipelined, in-order-pipelined, and out-of-order) have been synthesized, targeting the Bluesim simulator. Four simple benchmarks [88] were run to assess the simulation speed and processor's performance: the numeric median filter, multiplication, Towers of Hanoi, and vector-vector addition.

Performance of processor models

The performance evaluation of the processor is represented in Figure 2.7A. The performance of the unpipelined processor is constant because every instruction is always executed in one model cycle, giving a CPI (cycles per instruction) of 1. On our benchmarks, the in-order pipeline target achieves an average CPI of 2.6. The smaller CPI of the out-of-order processor, which enables the execution of more instructions per cycle, makes it faster than the in-order processor. The out-of-order processor performs best on the vector-vector add benchmark because its performance is based on the amount of available instruction-level parallelism.

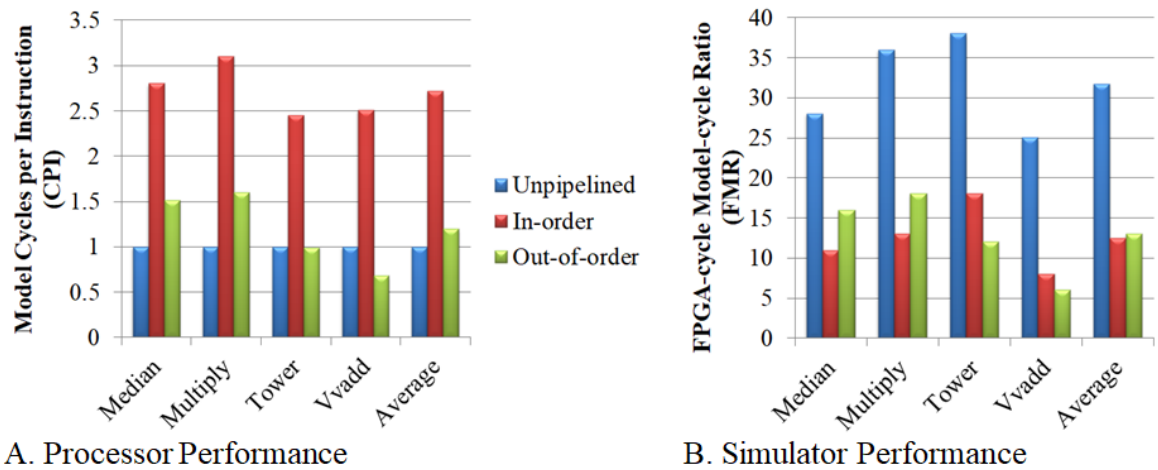


Figure 2.7 – Evaluating the performance of the simulators with target processsor models.

Performance of simulators

The principle behind using FPGA for processor simulation is that one tick of the FPGA clock does not match one tick of the target model's clock. This enables the processor's components to be simulated in parallel on available areas of the FPGA, while a separate mechanism ensures that their simulated timings fit the design objective. We don't need to configure the FPGA in the hardware model. Alternatively, we need to have a mechanism that precisely counts the number of clock cycles in the target model. We will use the FPGA-cycles-to-Model-cycles Ratio (FMR) [90] to analyze the simulation speed. FMR is expressed as the average ratio of FPGA cycles to model (i.e., target processor) cycles. We will have to consider that the higher the FMR is, the less time is necessary to simulate.

FPGA-cycles-to-Model-cycles Ratio

$$FMR = \frac{cycles_{FPGA}}{cycles_{model}} \tag{2.1}$$

The evaluation of the simulators' performance for each considered architecture is shown in Figure 2.7B. The unpipelined model has the slowest simulation rate, while the 5-stage model achieves the fastest, with the out-of-order model in the middle. Because

the timing model performs each of the seven functional partition operations for an instruction before even beginning to fetch the next one, it should not come as a surprise that the unpipelined simulator is so slow. This results in 31 FPGA cycles being spent replicating one model cycle. Timing-directed simulation is not suitable for this objective because the model cannot exploit the parallelism that is present in the functional partition.

In contrast, the in-order pipeline simulator is absolutely faster than the unpipelined processor simulator, taking an average of 12.5 FPGA cycles to simulate one model cycle. This is due to two reasons: firstly, the pipelined architecture of the model means that it executes functional partition operations in parallel; and secondly, the fact that the target circuit delays the pipeline for back-to-back dependent instructions actually increases the simulation rate. Both of these considerations contribute to this result. Because they do not need invocations of the functional partition, pipeline bubbles may be simulated very quickly. This makes them very efficient.

It is difficult to evaluate the out-of-order simulator. It has a slower FMR ratio than an in-order pipeline. In this instance, the simulator's limiting step is dictated by the timing model itself. The out-of-order processor's performance is reliant on the amount of instruction-level parallelism available and consequently performs best on the vector-vector addition benchmark. The usefulness of the out-of-order simulator is significantly benchmark-dependent. Due to the time necessary to multiplex the ALU during each model cycle, the out-of-order simulation in Vvadd and Tower applications is the slowest.

2.3.4 Targeting the Xilinx Virtex-7 FPGA platform

We designed three different processor models (non-pipelined, pipelined, and out-of-order) and synthesized them to target the Virtex-7 FPGA. The VC707 [8] is an evaluation board that is extensively used and was designed using the Virtex-7 LX485T. The VC707 FPGA, which communicates with a host computer over a high-speed peripheral component interconnect express (PCIe) connection, is the board that will be the focus of this discussion. The VC707 has 1 gigabyte (GB) of DDR2 onboard memory. Table 2.3 summarizes the synthesis results in terms of FPGA slices and block RAMs.

LEAP supports VC707 evaluation board using BlueNoc Linux drivers. We rein-

8. https://www.xilinx.com/products/boards-and-kits/ek-v7-vc707-g.html#overview

	Unpipelined	In-order-pipelined	Out-of-order
FPGA slices	87193 (28.7%)	135890 (44.75%)	215703(71%)
Block RAMs	65 (6.35%)	83 (8.05%)	85(9%)
Clock Speed	100 MHZ	100 MHZ	100 MHZ

Table 2.3 – Synthesis results for a Virtex-7 VC707 FPGA platform.

stalled the LEAP base platform tools and BlueNoc [9] Linux drivers to target the VC707 FPGA. Users have to configure an environment by running a provided "settings file," resulting in poor compatibility with other tools in the LEAP infrastructure. Unfortunately, the driver that is supposed to connect the host computer and the FPGA is not working properly. PCIe device administration in LEAP needs knowledge of system addressing. It is necessary to examine the different components of the system to determine the PCIe address of the hot-plug controller for a certain card. Moreover, after the FPGA has been programmed, the system must be rebooted. Host computers often require a restart since their PCIe controller hardware lacks hot-plugging capabilities.

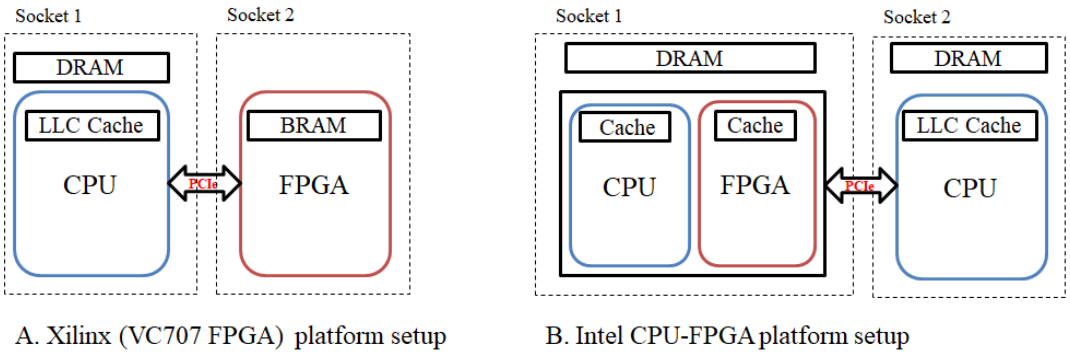


Figure 2.8 – FPGA-based platform overall setup.

In our study, simulation models are implemented on a platform that includes both the CPU and the FPGA. Device locality and fast interconnects between the host computer and FPGA are critical for accelerating simulations. Regardless of the interface employed in CPU-FPGA platforms, data transmission has an impact on simulation speed's communication delays. As depicted in Figure 2.8A, a typical FPGA-accelerated server consists of cards in a PCIe slot on the motherboard of the host server. This configuration typically results in latency. In addition, the complexity of the architectural design

9. https://github.com/LEAP-FPGA/leap-documentation/wiki/ML605-and-VC707

target can result in additional overhead in situations where the data stored on the CPU needs to be transferred to the FPGA.

Intel's Broadwell Xeon multicore processor with integrated Arria 10 FPGA capabilities is motivated by communication delays within hybrid architectures. Intel places the FPGA close to the CPU (reducing the distance from inches to millimeters) and also connects them via a high-speed QPI link. Intel also intends to characterize the communication delays between CPUs and the FPGA by utilizing both the low-latency cache coherent interface and the two PCIe links provided by this platform. The Intel Xeon CPU with FPGAs is considered to deploy our simulation models.

2.4 Deploying the HAsim simulator on a Intel CPU-FPGA platform

HAsim is a hybrid simulator that employs both a central processing unit (CPU) and a field-programmable gate array (FPGA). We can utilize the FPGA for fine-grained parallelism and the CPU for infrequent but challenging-to-implement events that may appear in the simulator, such as system calls. While FPGA-based simulators offer significant speedups over software-based alternatives, they are not without limitations. The latency that occurs during the transfer of large amounts of data between the FPGA and CPU can be considered a problem that slows simulation speed. In addition, a large number of operations wait for the FPGA to read and write host memory. Reducing the latency of data transfer between the FPGA and CPU, which includes speeding up the communication channel and shared-memory access, is one of the most important strategies for accelerating the FPGA-based simulator. The latency-insensitive environment for application programming (LEAP) is mostly focused on latency-insensitive communication channels by providing a large number of portable abstraction layers for program development. Communication delays within hybrid architectures, on the other hand, have gotten much less attention. A solution proposed by Intel using Intel's Broadwell Xeon multicore processor with integrated Arria 10 FPGA capabilities appears to be an interesting approach to reducing communication delays between CPUs and the FPGA, utilizing both the low latency cache coherence interface and the PCIe links provided by this architecture.

2.4.1 Intel Xeon+FPGA platforms

Through the IL Academic Compute Environment [10], academics may have access to Intel technology, which will hopefully advance research efforts. It offers two separate FPGA-CPU solutions with the Intel Arria 10 FPGA: an integrated CPU and FPGA and a programmable acceleration card (PAC). The innovative hardware solution that shares and adapts the OPAE application programming interface (API) and CCI-P used for FPGA-CPU communication is made possible by Intel CPU-FPGA systems. This design is intended for large workloads in data centers as well as application-specific hardware.

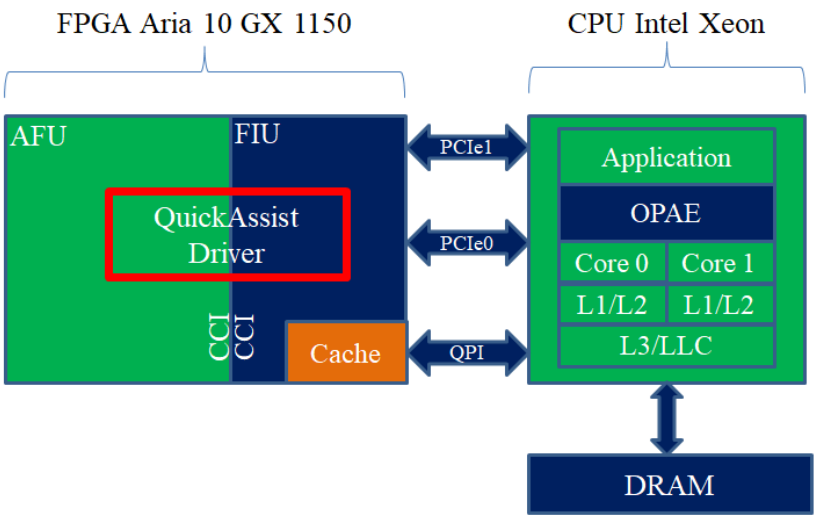


Figure 2.9 – An overview of the architecture and hardware of Intel Arria systems. The "Green Region" identifies the portion of the FPGA that may be reconfigured in user space during runtime. The "Blue Region" describes the FPGA's static soft core (Intel API). It makes the CCI-P interface accessible to the AFU.

Figure 2.9 provides an overview of the components of the BDX platform. The FPGA is divided into two distinct sections. The blue region, which is FPGA interface unit (FIU), is provided by Intel and does not undergo any changes during runtime. The Green Region, which is called the accelerated function unit (AFU), is capable of being reconfigured by users. The CPU part is connected to the FPGA through the QuickPath interconnect (QPI), a point-to-point processor interconnect that increases scalability and available bandwidth (it has a speed of 6,400 gigabits per second). The fact that the FPGA and CPU share the same address space minimizes overhead by eliminating

10. https://wiki.intel-research.net/index.html

memory transfers between the CPU and FPGA. Using a cache protocol, the CPU and FPGA interact with one another. The CPU can deliver 20 GB/s of bandwidth. AFUs can reach the last level cache of the CPU. In an ideal situation, we would acquire a read hit and not have to access system memory.

The Open Programmable Acceleration Engine, often known as OPAE, is a software framework that was developed in order to manage and access FPGAs. Developers may use the OPAE Software Development Kit (SDK) [11] to build AFUs as well as Linux drivers that work with both PACs and BDXs. To transfer data from the CPU to the FPGA, the API uses two methods: memory-mapped I/O (MMIO) and direct memory access (DMA). Software programs have the ability to submit MMIO requests to the AFU with a width of either 64 bits or, optionally, 32 bits. It is also possible to give DMA to an AFU in certain sections of the system's main memory.

Core Cache Interface (CCI-P) [57] is a host interface bus for an AFU that consists of separate wires for the header and the data. It is designed to connect an AFU to a FIU within the FPGA. It saves a lot of time and effort by giving MMIO and DMA requests a simple, hardware-independent way to be handled. CCI-P provides an abstraction layer that may be applied on top of various platform interfaces, such as PCIe and QPI. Therefore, AFUs developed for CCI-P can be synthesized for any CCI-P-compatible platform without requiring design modifications. The CCI-P is an architecture for developing reusable FPGA libraries consisting of hardware and software modules and may be viewed as the "API" for developers of accelerators.

2.4.2 Implementing communication channels support for Intel CPU-FPGA platform

Figure 2.9 illustrates the QuickAssist Xeon+FPGA platform. This platform is comprised of a multicore Xeon processor and an Arria 10 FPGA that share memory access via a variety of physical channels, including Intel's Quick Path Interconnect (QPI) and two PCIe 3.0 x8 links. In order to construct the communication channel between the host CPU and the FPGA, we made use of the QuickAssist (QA) driver that we had extended [12]. This work is summarized in Figure 2.10. The purposes of the QA driver are:

11. https://github.com/OPAE/opae-sdk
12. https://github.com/LEAP-Core/leap-platforms-intel

— interface to a host/FPGA channel with the same width as a cache line and a simple First In First Out (FIFO).

— interface to system memory for read/write requests.

— interface to the CCI-P provided by Intel.

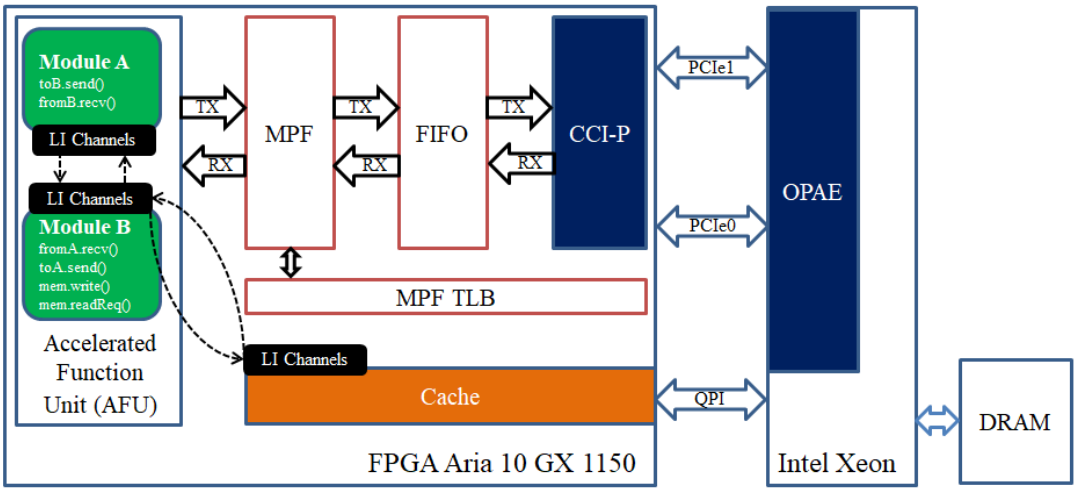


Figure 2.10 – Quick Assist (QA) driver with CCI-P and OPAE.

In order to interface with physical devices, software operating systems employ hardware abstraction layers. Each device provides a standard API that is shared by all devices of its type. This strategy is embraced by LEAP. Classes of physical devices provide a standardized, abstract device interface in LEAP. As shown in Figure 2.11, LEAP devices offer a module-based FPGA environment (*fpgaenv*) instead of a call-based API. This environment has a virtual platform with a set of hardware driver layers that provide a uniform, abstract device interface. FPGA platforms can be viewed as a collection of low-level device driver modules. LEAP accepts a platform description file containing abstract drivers for each physical device on each target platform. We extended drivers that are compatible with the Intel CPU/FPGA platform, which can be found in Figure 2.11.

A connection can be made between the AFU and the host CPU through the use of CCI-P. By providing a specific memory address, an AFU is able to obtain data from the system's memory. An AFU is also able to write data to the system memory by providing a specific address to write to as well as the data that is to be written.

Unpipelined RV64I Model - HARPv2

Type	Flag	Implementation
model	n	HW/SW Hybrid Pipeline Build
application_env	n	Soft Services Hybrid Application Environment
connected_application		HAsim Performance Model
hasim_common		HAsim Common Default Configuration
hasim_funcp		RISCV/RV64I Functional Model V2 (gem5)
hasim_isa		RISCV/RV64I ISA Definition - pipelined
hasim_model_services		Default HAsim Model Services
hasim_timep		Single Chip Timing Partition
hasim_chip		Single Core Chip
chip_base_types		Chip Base Types
hasim_chip_topology		Standard Topology Manager
hasim_core		Cacheless Core
hasim_pipeline		Hasim Unpipelined Pipeline
hasim_memory		Null Memory
memory_base_types		Memory Base Types
hasim_modellib		HAsim Modeling Library
hasim_cache_algorithms		Cache Modeling Algorithms
fpgaenv		Intel QuickAssist CCI-P FPGA BDX Xeon+FPGA
project_common		Default Hybrid Project Common Utilities

Intel QuickAssist CCI-P FPGA BDX Xeon+FPGA

Type	Flag	Implementation
fpgaenv	n	FPGA Environment
build_pipeline		FPGA Build Pipeline Altera for QuickAssist CCI-P BDX
fpga_components	p	Hardware FPGA Components for StratixIII
librl_bsv		RL Bluespec Library Submodel
soft_services		Soft Services Submodel
virtual_platform	n	Standard Virtual Platform

Intel QuickAssist CCI-P FPGA BDX Xeon+FPGA

Type	Flag	Implementation
soft_services		Soft Services Submodel
virtual_platform	n	Standard Virtual Platform
low_level_platform_interface	n	Low Level Platform Interface Hybrid
channelio	n	Basic Virtual Channel Multiplexer
physical_channel	n	Intel QuickAssist FPGA Physical Channel
local_mem		Local Memory interface using QA host memory
local_mem_interface	n	Local Memory Interface
physical_platform	p	Intel QuickAssist Platform
clocks_device	pn	Dual Frequency Provided Clocks Device
ddr_sdram_device	n	Null DDR SDRAM Physical Device
ddr_sdram_definitions	n	DDR SDRAM Common Definitions
physical_platform_utils		Common Physical Platform Utilities
physical_platform_defs		Common Physical Platform Definitions
qa_device	n	Intel QuickAssist Device
qa_driver		Intel QuickAssist Driver
qa_cci_mpf	n	Intel MPF (Memory Protocol Factory)
qa_cci_mpf_hw		Intel QuickAssist MPF HW Library
qa_cci_mpf_sw		Intel MPF SW
qa_cci_mpf_sw_cxx_include		Intel MPF SW CXX Include
qa_cci_mpf_sw_include		Intel MPF SW Include
qa_driver_host_channels		Intel QuickAssist Host/FPGA Channels
qa_platform_libs		Intel QuickAssist CCI-P Platform Libraries
qa_ccip_async_hw		Intel CCI-P Edge Clock Crossing FIFOs
qa_ccip_async_hw_par		Intel CCI-P Edge Clock Crossing FIFO PAR Files
physical_platform_debugger	n	Intel QuickAssist Platform Debugger
remote_memory	n	Null Remote Memory
rrr	n	RRR Clients and Servers
rrr_common	n	RRR Common Module
rrr_debug	n	RRR Debug Module
platform_services	p	Standard Platform Services (Local Memory Scratchpad) Submodel
umf	p	Little-Endian Unified Message Format
virtual_devices		Standard Virtual Devices Submodel

Quick Assist Driver

Figure 2.11 – An overview of the leap environment includes the QA driver modules for the Intel CPU-FPGA platform.

A standardized set of memory semantic extensions for CCI is made available via the Memory Properties Factory (MPF). It has a number of logic layers that define how to access memory. For instance, the virtual to physical address translation (VTP) layer offers support for virtual address translation, which enables the AFU to do read/write requests by making use of virtual addresses. In a similar manner, the read ordering (RO) layer supports read response ordering, which ensures that responses are received in the same order as requests are given by the AFU.

On the host CPU, software that uses OPAE can access an AFU. OPAE is responsible for defining the protocols that are necessary to initialize modules on the FPGA, creating a shared virtual address space for the CPU and FPGA, and communicating with AFU. Specifically, OPAE is responsible for the creation of, as well as the provision of, procedures for accessing a collection of control and status registers (CSRs). These are what are used to begin the operation of an AFU, to transmit the AFU's parameters, and to otherwise connect with and control the AFU. To transmit a message, the host sends data to the FPGA's CSRs via the channel interface. Once a message has been completely assembled in CSRs, the FPGA logic handling the connection stores the message in a FIFO within the FPGA logic. The FPGA module on the channel's receiving side retrieves new messages from this FIFO. In a similar way, an FPGA module sends a message to the host on the transmit side of its channel, where it is then stored in another on-chip FIFO. The host draws from this FIFO through successive MMIO reads to CSRs after verifying a status flag indicating a new message is available.

2.4.3 Validation

We used several benchmarks of significantly varying sizes in order to demonstrate how LEAP is validated on the Intel CPU-FPGA platform. Table 2.4 summarizes the results of experimental application synthesis for the Arria 10 FPGA.

Hello World: This program, much like its well-known equivalent in the software world, first prints out a message and then terminates.

Counter: In this simple HW/SW hybrid application, a counter is implemented in hardware utilizing shared memory, and the streaming device is used to show the count and status messages from both HW and SW.

FPGA-Host Channels: Communication between the host CPU and FPGA in an Intel CPU/FPGA environment is crucial and frequently more difficult. The primary fo-

cus of test programs is on the communication channels between the FPGA module and the send/receive stream software running on the host CPU. We send a stream of data from the host CPU to the FPGA and receive a stream of data generated by the FPGA. Additionally, we send numerous data packets in both directions and validate the data during the transfer. The results, as well as the number of error packets and their bandwidth, are included in the reports.

HAsim: As stated in section 2.2.1, HAsim is a framework for the construction of high-speed and cycle-accurate simulators of processors [87]. HAsim recycles parts of a single processor and distributes them to several modeled processors so that they can be used more than once. Both in terms of its structure and the number of cores that it models, HAsim features a high level of parametric flexibility. Because of its scalable architecture and parameterization, HAsim models are able to scale to hundreds or thousands of cores with just a few changes to the source code. This, however, is predicated on the assumption that the operating system has sufficient support to map large designs. The target architecture is the RISC-V unpipelined processor model.

	LUTS	**Registers**	**RAM Blocks**	**Memory Bits**
Hello World	89,172(21%)	108309	398(15%)	2,975,536(5%)
Counter	114,810(27%)	148333	542(20%)	4,617,402(8%)
Channel-16	95,012(22%)	117990	409(15%)	3,297,037(6%)
Channel-32	97,479(23%)	122529	409(15%)	3,297,037(6%)
Unpipelined Processor	130,113(30%)	160115	600(22%	3,954,350(7%)

Table 2.4 – Results of experimental application synthesis targeting Arria 10 FPGA.

Size of data	1 GB of data		100000 packets
	Latency(s)	Bandwidth(GB/s)	Error packets
Host –> FPGA	0.2	4.910	1-700
FPGA –> Host	0.266	3.759	0
Host –> FPGA –> Host	0.35	5.642	–

Table 2.5 – CPU-FPGA communication channel validated in Intel Xeon+FPGA.

The QA driver is step-by-step validated for operation through the development of test programs. Firstly, Hello World and Counter can be executed successfully to demonstrate that the configuration and communication between the CPU and FPGA are operating correctly. Secondly, the results of the FPGA-Host Channels test are illustrated in Table 2.5. It takes 0.2 seconds to transfer 1 GB of data in each direction, and 0.35

seconds for two directions. The bandwidth ranges from 3.7 to 5.6 GB/s. The channel transferred data successfully, but when we transferred 100000 packets of data from the host to the FPGA in various test runs, we obtained fluctuations from 1 to 700 error packets. This issue is a result of the failure of the last test to run HAsim to simulate an unpipelined processor. Due to the use of low-level simulation to debug, predicting the cause of error data during transfer requires a significant amount of effort. This is also one of the challenges in developing an FPGA-based simulation project, as discussed in the following section.

2.5 FPGA-Accelerated microarchitecture simulation challenges

HAsim FPGA-accelerated simulation is not widely utilized by computer architecture designers [13]. There are various technical explanations for this, including the perspective of simulator implementation. This section summarizes the challenges associated with using FPGAs to simulate SoCs that we faced during our research.

Controlling and utilizing various development tools

Platform development refers to the collection of tools, libraries, and hardware that lead to the construction of simulation models and the operation of an FPGA-based simulator. Figure 2.12 illustrates the various software tools required to construct simulation models, including:

— The LEAP version 15.02 [12], packages are available for Ubuntu 14.04 . LEAP is released under BSD and MIT license terms.

— AWB is the Asim Architect's Workbench [13], a set of abstractions that enables the plug and play of modules to facilitate design.

— The core libraries and utilites for using the Asim modeling infrastructure [14].

— Bluespec System Verilog(version Bluespec-2017.07.A) and Bluesim simulator. [15].

12. https://github.com/LEAP-Core/leap
13. https://github.com/AWB-Tools/awb/wiki
14. http://asim.csail.mit.edu/apt/releases/ubuntu
15. http://wiki.bluespec.com/Home/BSV-Documentation

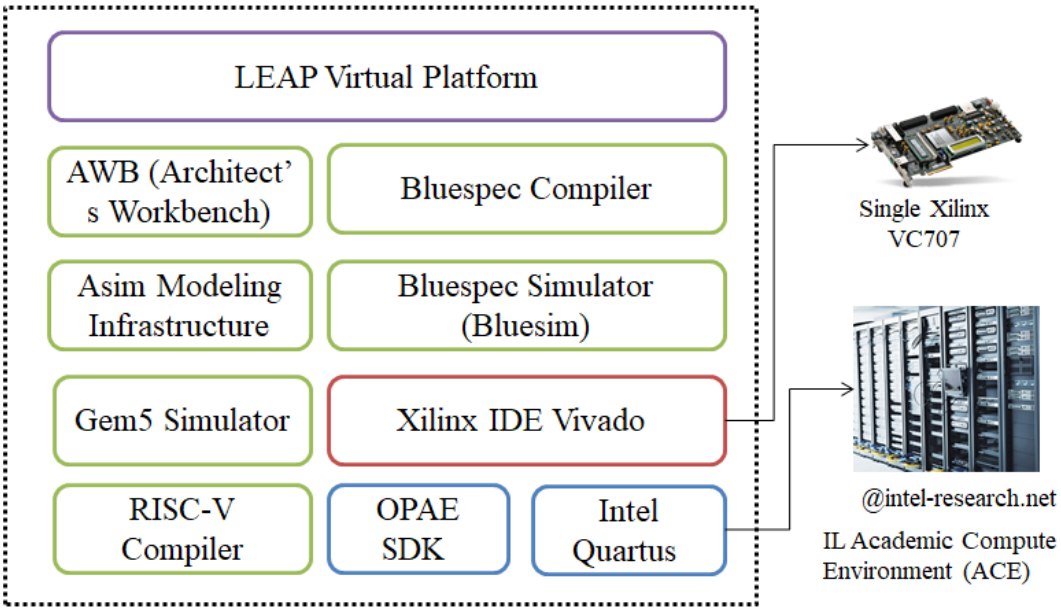


Figure 2.12 – Set of FPGA-based platform development tools.

— The RISC-V software toolchain [16]. It includes RISC-V compiler using for generate binary code when running benchmarks.

— The gem5 computer-system architecture simulator has been adapted to allow an interface with HAsim. It is also used to simulate the system-level architecture as well as processor microarchitecture, like the original gem5 [13] simulator.

— The Virtex-7 FPGA is targeted using Xilinx Vivado HLS version 2018.3.

— The Open Programmable Acceleration Engine (OPAE), a software layer, works with Quartus Prime Pro 17.1.1 to target servers and FPGAs with Intel Xeon processors. The experiments are done on the Illinois Academic Compute Environment (ACE) [14], which is a place for supporting and advancing academic research in many different areas.

— Programming languages like C/C++, Bluespec, Verilog, and scripting languages are used.

Software-based simulators are inexpensive and can run on desktop machines,

16. https://github.com/riscv/riscv-tools
13. https://www.gem5.org/
14. https://wiki.intel-research.net/Introduction.html

whereas FPGA-based simulators require expensive hardware. FPGAs range from a few euros to thousands of euros in price. It is important to choose an FPGA that works well with the application. If the application requires a large number of tasks to be processed in parallel, the cost of an FPGA is considered. FPGA-based simulators are used to run massive parallel experiments, which require the most powerful FPGA board, which means we have to buy expensive hardware. If the simulator utilized low-cost FPGA boards, simulation capacity would be reduced, and many resource optimization efforts would be required. Additionally, the partitioning of large designs across multiple FPGAs must be done manually or with the aid of specialized tools (such as HAsim project) because they cannot fit on a single FPGA. This increases the cost of the simulation based on FPGAs.

It is difficult to install the environment and reproduce the result when designers approach an FPGA-based simulation project. For instance, because FPGA-based simulators are dependent on particular FPGA platforms, researchers are required to get the same host in order to reproduce published results. Even if researchers had access to their FPGAs, they could not usually run existing simulation models without modifying them.

Complexity of simulation modeling

Designing these models in RAMP required less time and effort as compared to writing RTL for the actual implementation. However, RAMP simulator designers stated that designing models was more challenging than writing RTL for the corresponding implementation. For example, consider that when modeling the detailed, cycle-accurate behavior of this pipeline processor, the model designer must write RTL that contains a significant amount of the complexity inherent in the actual pipeline's design in order to capture all hazards that may affect the processor's performance. In addition, designers must add even more complexity to support modeling a space of different processor designs, either at compile time by generating different model RTL or at runtime by adding logic to allow reconfiguration of the simulator. Figure 2.13 captures the structure of the model in the HAsim framework; it is a tree structure with many branches. For each branch, modules are associated, written in Bluespec. Three main categories make up our division:

— The functional partition and instruction set definition (*hasim_funcp*, *hasim_isa*)

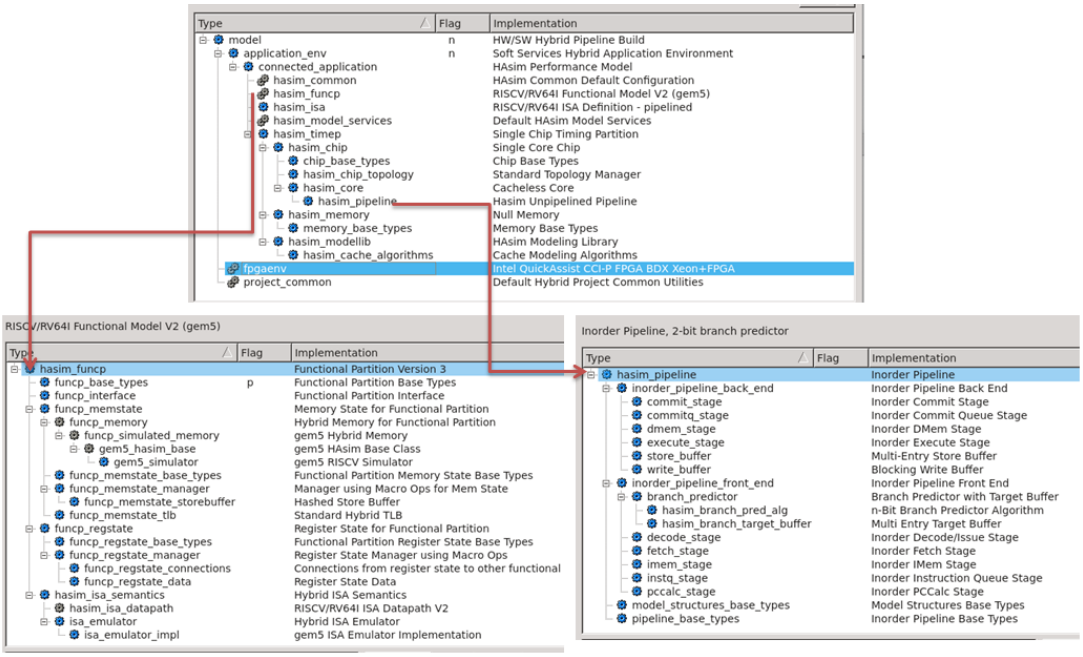


Figure 2.13 – An overview of modules to develop functional and timing partitions in in-order pipelined processor models.

are responsible for correct ISA level execution of the instruction stream. This part is written once and can be reused by many timing models.

— The timing partition (*hasim_timep*) is responsible for tracking micro-architecture specific timing, such as branch predictors and cache misses. This component is changed according to the chosen timing model (unpipelined, in-order-pipelined, and out-of-order). As shown in figure 2.13, it is an example of in-order-pipelined simulation, where stages of the pipeline are integrated in *hasim_timep*.

— Another set of modules in the HAsim framework (*hasim_common*, *hasim_model_services*, *fpgaenv*) are not directly related to simulation. They provide common utilities, such as tracking statistics and parameters and interacting with the host CPU as well as the virtual platform.

Development efforts

HAsim's partitioning scheme can remarkably reduce development time because the same functional partition can be reused across different timing models. Table 2.6 illustrates the lines of code required to implement each partition as well as the reusability of

the code. The timing model does not need to implement all structures because some of their functionality can be reused when developing different timing models, reducing developer effort. For example, it was possible to use the same code for both the in-order and out-of-order models of timing. First, the whole functional partition was used again without any changes. In the timing partition, the branch predictor was the most likely to be used again, and it was used exactly the same way. The designers have to write a lot of code, but they don't have to re-implement the functionality of the instruction set for every new simulator. If they change the processor target, only its partition-related ISA will be implemented. When targeting a new architecture and simulation hardware platform, development and debugging efforts are still necessary.

Category	**Lines of Bluespec Code**	**Code Reuse**
Functional Partition	3133	All modules
RISC-V ISA	691	None
Unpipelined	405	Some modules
In-order-pipelined	725	Some modules
Out-of-order	1219	Some modules
Communication Channel	316	None

Table 2.6 – Lines of Bluespec code to implement the simulation models.

The amount of time and effort required to develop fpga-accelerated simulators is an important factor to consider. FPGA-accelerated simulators may fail not because of simulator performance but because the increased development time makes modeling take longer than with slower software. Architectural simulators are built in software, which reduces development time and enables small teams of skilled architects to conduct computerized architectural studies. Developing fpga-based simulators currently necessitates greater development efforts than software for the reasons outlined below:

— Insufficient design visibility makes it challenging to debug failing systems. There is no standard library infrastructure compared to software, which makes printout-based debugging more difficult. Designers must instantiate fpga-specific debugging modules. The design must be resynthesized with new tested inputs if the bug is not found on the first iteration.

— Prolonged development times are made more difficult by long compile and synthesis times (1 to 10 hours), which increase the time spent on the debugging step. To do a test, we have to perform multiple iterations of the synthesis process and determine whether the design meets the constraints set at each stage. For a reasonably large design, this can take a lot of time.

— For fpga-accelerated simulators, the development time is a major concern. The designer must initially implement and validate simulation models on Bluespec. Additionally, the models must be compiled for the platform using numerous tools and libraries. Therefore, platform compatibility has a twofold effect on development time. All structures must be implemented, tested, and ported, resulting in an increase in developer effort.

2.6 Conclusions

Most computer system simulators are implemented with software because they need to be fast and flexible. Unfortunately, given the rapid growth of their complexity over time, software simulators are slower, and the performance of computer simulation has decreased. The use of hardware that directly matches the hardware level parallelism required in accurate computer system simulation is a fast and cost-effective alternative to software. FPGAs are an ideal vehicle for accelerating and addressing the challenge of computer system simulation because of their significant capabilities and flexibility.

HAsim does not reflect the simulation of any particular target CPU running on any particular FPGA platform. Instead, HAsim is a general framework that can be used to model target processors with a wide variety of properties and then run those models on FPGA platforms. Developing for FPGAs continues to be much more difficult than developing software; therefore, HAsim places a significant emphasis on simplifying development efforts.

We were able to construct three different RISC-V processor models by utilizing HAsim framework (unpipelined, in-order-pipelined, and out-of-order). We also demonstrated the implementation of communication channels on systems that tightly couple a CPU and FPGA. Using the Intel QuickAssist Xeon+FPGA platform as a cloud platform, we aim to accelerate and scale the HAsim simulator in order to design more complex heterogeneous systems.

To summarize, while HAsim can be fast and accurate for simulation, building complex SoC architecture models onto an FPGA is still challenging. There are difficulties in approaching the complexity of simulation modeling and development efforts. Due to these factors, FPGA-based microarchitecture simulation may not be utilized by the majority of SoC designers.

CHAPTER 3

INTEGRATION OF A PRE-RTL ACCELERATOR MODEL IN THE FPGA-BASED SIMULATOR

Currently, computer architects are focusing on heterogeneous multi core architectures combined with special-purpose accelerators that can improve performance and reduce energy consumption by orders of magnitude. However, because of their inherent complexity, heterogeneous systems are difficult to design. Building tools to support these architectures has been a growing topic in both academic research and commercial development for the past decade. In this chapter, we present a method for developing heterogeneous systems using a combination of accelerator and processor models relying on an FPGA-based simulator. Our approach takes high-level language descriptions of application-specific programs as inputs and models an accelerator using dynamic data dependency graphs (DDDG) without generating RTL. The accelerator model is integrated into the HAsim simulator, and the processor models from Chapter 2 are used. Initially, we illustrate the design flow and the steps of the proposed design methodology. The accelerator modeling methodology is then presented, which is based on the Aladdin simulator. After that, by combining the accelerator and processor models, we will demonstrate how to use HAsim to simulate heterogeneous systems. Finally, experiments for a subset of the MachSuite benchmark [98] are presented and their results discussed. As a result, the design of application-specific heterogeneous architectures is simulated by an FPGA simulator at an early stage of the design.

Contents

3.1 Introduction **74**
3.2 Design flow overview **76**
3.3 Accelerator modeling (Pre-RTL accelerator model) **78**
3.3.1 DDDG generation and scheduling 78
3.3.2 Scheduled Graph Trace (SGT) generation 79
3.3.3 Flow explanation by an example 81
3.4 Integration of an accelerator model in the HAsim simulator **82**
3.5 Performance assessment **86**
3.5.1 Case study: Blocked Matrix Multiply accelerator 87
3.5.2 Machsuite benchmarks 89
3.6 Conclusion **90**

3.1 Introduction

Nowadays, artificial intelligence and data science are among the most emerging technologies that demand more computation power from hardware. Several highly demanding applications currently include these algorithms, such as self-driving vehicles, 5G communication, video monitoring, and analytics. In an effort to support these applications on vehicles and other mobile devices, scientists and researchers focus on embedded high performance computing architectures to solve these problems effectively and quickly.

To address these significant computational demands, some companies have already proposed solutions based on processor core duplication and pushed the number of cores on a chip as high as a thousand [15]. However, the number of cores in these multi/many-core chips that can actively switch at full speed within the chip power budget will decrease (utilization wall). The remaining silicon, which is left unpowered (referred to as "dark silicon"), will grow exponentially with each new technology generation [42]. To address this, heterogeneous architectures combining processor cores and hardware accelerators [28, 73] have been proposed by designing heterogeneous systems that trade dark general purpose cores for a collection of specialized but transiently powered accelerators. Architectures with customized accelerators induce orders-of-magnitude performance and energy efficiency improvements compared to performing the same task on a general-purpose CPU.

In Chapter 2, we exploited the fine-grained/coarse-grained parallelism of FPGA to speed up the simulation of computer architecture. However, these techniques rely on execution-driven models, which consume an excessive amount of FPGA resources and make it more difficult to model various types of custom hardware. Numerous researchers have already described the use of trace-driven simulation of multi-threaded programs on multi-core platforms. Using trace-driven simulation has several advantages. Interestingly, it does not need the functional execution of the original code, which makes it perfect for accelerator simulations where an execution model would take excessive resources. Also, a trace is only made once and can be used for many simulation-based architectural explorations. These simulations may streamline the design, saving FPGA resources and development time while also accelerating simulation speed.

In this chapter, we present a design approach for application-specific heteroge-

neous architectures based on the usage of performance models of hardware components (processor cores, domain-specific accelerators, memories, and interconnects). The simulator of a particular heterogeneous architecture can be built from these models and deployed on an FPGA-based system, thus helping to speed up the simulation. It is a question of taking into account data movement between hardware components and coherency management for accelerators during the simulation. This aspect is often neglected, and its impact has however been well shown by Shao et al. in [105].

We are particularly interested in how the accelerator models are designed, starting from the C code of the algorithm and showing how they are integrated with the other components of the heterogeneous architecture. The data flow graph of the algorithm is used as a representation of the accelerator without having to generate HDL (Hardware Description Language) code. From this representation, the scheduling of the graph is realized, and the control for a generic template of an accelerator model is produced. The innovative technique for generating traces brings the trace size down by several orders of magnitude, making it possible for the majority of the trace to be stored in the embedded RAMs of the FPGA. This, in turn, speeds up the simulation while reducing the amount of communication needed with the host CPU.

The proposed simulation infrastructure is based on the use of two simulation tools: Aladdin [105] and HAsim [87]. The Aladdin accelerator simulator provides a framework for modeling the power, performance, and cycle-level activity of standalone fixed-function accelerators without the need to generate RTL. The HAsim FPGA-based simulator allows the building of processor timing models and memory systems, including support for cache coherence protocols and interconnect models. The contributions presented in this work are as follows:

— A methodology for generating performance models of accelerators targeting the FPGA by exploiting a part of Aladdin flow is proposed. The application trace is generated in a compact graph scheduled trace (SGT) format. It can be directly interpreted by the timing accelerator model on the FPGA to re-create the original computation and memory behaviors of the application.

— A cycle accurate and cost-effective simulation framework is presented. It simulates the whole system of the accelerator-processor architecture, including the RISC-V core, dedicated accelerators, coherent cache/scratchpad with shared memory, and routers. The framework targets platforms combining CPU and FPGA with the goal of speeding up the simulation.

3.2 Design flow overview

Our approach to designing domain-specific heterogeneous architectures is based on a trace-based accelerator simulator that profiles the dynamic execution of a program by integrating hardware accelerators coupled to a single-core processor. The starting point is the construction of a dynamic data dependence graph (DDDG) as a data flow representation of an accelerator. Fig. 3.1 shows the generic flow of the proposed design method. It consists of the following steps, which are described in the following sections:

— Application development utilizing the C/C++ programming language to analyze systems within a given domain and develop specialized hardware blocks.

— Modeling of the accelerator using dynamic data dependency graphs (DDDG) without RTL generation(3.3).

— System integration where the hardware accelerator is integrated with processor models (3.4).

— Simulation where all models are simulated and evaluated on the FPGA-based simulator (3.4).

The design method aims to build an efficient architecture for a domain-specific application by integrating task-specific custom hardware into a single core. It takes as input a high-level specification (C/C++) of an algorithm without any modification and generates its execution trace. According to the designer's optimization pragmas (loop unrolling, loop pipelining, and array partitioning), a sub-trace is extracted, and the corresponding DDDG is generated. After applying a number of optimizations to the DDDG, the tool schedules the graph's nodes with resource constraints to obtain an early performance estimate of the accelerator. The obtained results can be exploited by hardware architects in order to further develop the application. By analyzing applications within a domain, designers desired to develop specialized hardware blocks to execute compute-intensive parts of these applications and integrate them with processor cores. When an application is executed on a processor, only the required accelerators are utilized in order to save energy.

The application software developer writes programs for a certain architecture using software development tools. Understanding how the development tools generate code and how to use them effectively is essential for producing code that achieves the desired outcomes. Even though the compiler will generate code optimized for a spe-

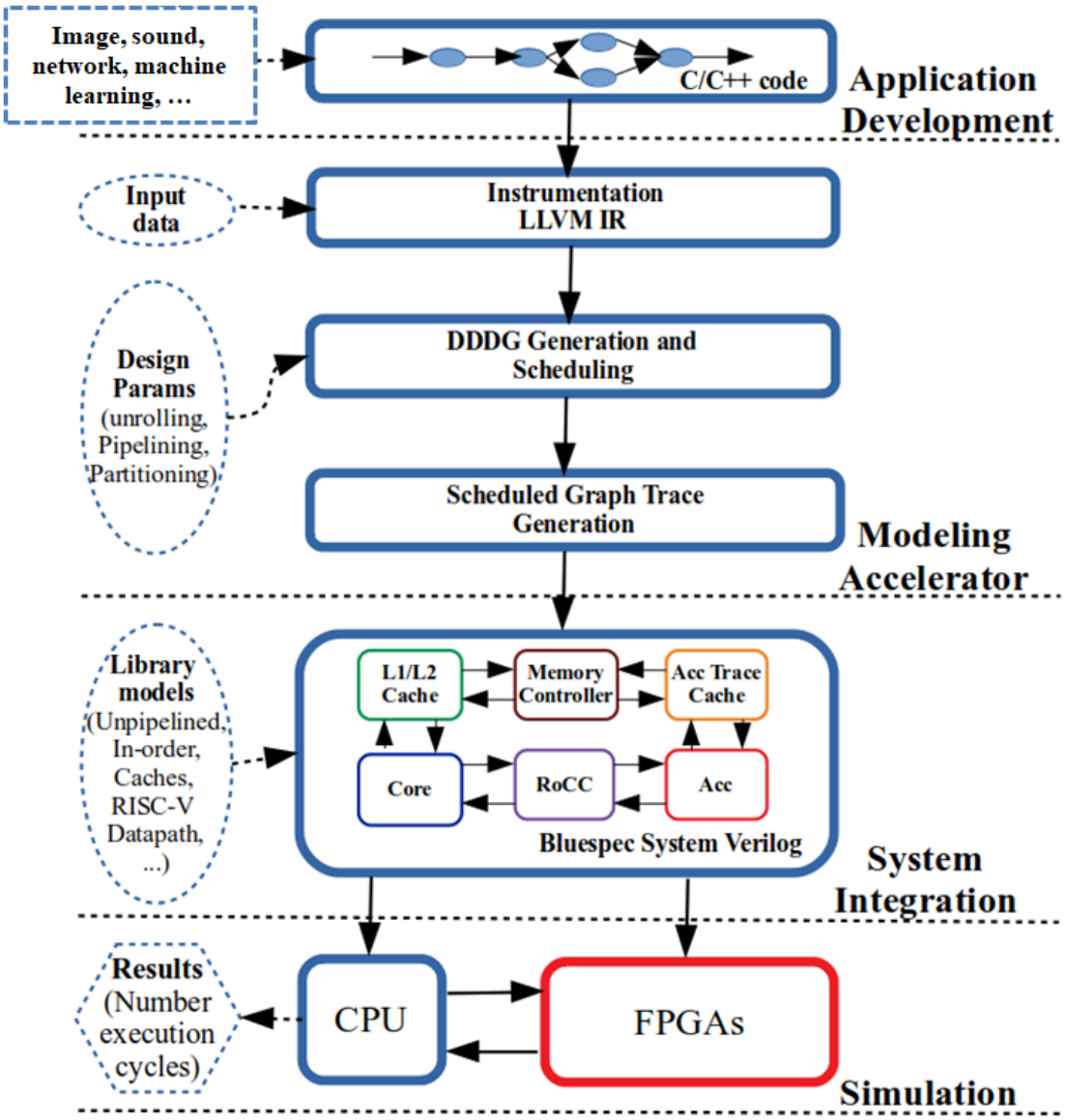


Figure 3.1 – Illustration of the generic design flow.

cific architecture, the developer must typically supply information to aid the compiler in producing optimal code. For instance, when we implement matrix multiplication algorithms, we can observe that the majority of calculations occur within the loop, so we will optimize the code by focusing on the loop. In certain applications, such as vector dot products, when dual MACs are desired on target, the loop is unrolled by a factor of 4 to maximize memory access bandwidth.

3.3 Accelerator modeling (Pre-RTL accelerator model)

We present a flow that accepts high-level language descriptions of algorithms as inputs and represents an accelerator using dynamic data dependence graphs (DDDG) without creating RTL. The tool starts with the unconstrained program DDDG, which represents accelerator hardware, and optimizes and constrains the graph to build a realistic model of accelerator activity. The application trace is then generated in a compact graph scheduled trace (SGT) format. It can be immediately understood that the timing accelerator model on the FPGA will reproduce the application's original computation and memory behaviors. In addition, we provide an example to illustrate the workflow of our tool.

3.3.1 DDDG generation and scheduling

Our accelerator modeling methodology is based on the Aladdin simulator to which modifications have been brought [105]. To define the accelerator modeling phases, we start from a C description of an algorithm before passing through an instrumentation phase where an execution trace of the application is generated. In this process, the low-level virtual machine (LLVM) [64] is leveraged for instrumentation and trace collection. The core of LLVM is a static single assignment (SSA) based intermediate representation (IR). The IR is machine-independent and uses unlimited virtual registers. The Execution Engine, an LLVM just-in-time (JIT) compiler integrated into the simulator, is invoked to execute the instrumented IR and generate the runtime trace. The generated trace contains runtime instances of static instructions, including instruction IDs, opcodes, operands, virtual register IDs, memory addresses (for load/store instructions), and basic block IDs.

After the instrumentation process, a DDDG is produced as a directed and acyclic graph, where nodes represent dynamic instances of LLVM IR instructions and edges denote dependencies between nodes, including register/memory dependencies. Edges only correspond to true dependencies and do not include output dependencies. These are dynamic traces, so control dependencies are not concerned.

Before scheduling, the DDDG is optimized by performing tree-height reduction to decrease long-expression chains' height and expose potential parallelism, similar to Shao's work [105]. Focusing only on actual computations, we remove supporting instructions (data movement and conversion between registers) and dependencies between loop index variables that are not relevant to accelerators. They are assigned zero latency to the associated nodes. Removing redundant load/store operations is also considered an efficient way to save memory bandwidth. For instance, two load operations can be reduced to one load node if they have the same memory address, so there is no store operation in between with the same address. The load can be eliminated by adding edges between the store and successors of the load if a store is a direct predecessor of a load with the same memory address. We map the memory address of load/store operations to a memory bank according to the array partitioning factor after removing redundancies. The graph is then scheduled for execution through a breadth-first traversal while considering user-defined hardware parameters: loop unrolling, loop pipelining, and memory ports.

3.3.2 Scheduled Graph Trace (SGT) generation

Using trace-driven, HW simulations on FPGAs instead of execution-based simulations avoids the need for resource-intensive execution units like floating point units. It saves FPGA resources, so multiple accelerator simulations can be run at once without having to time multiplex. Most execution events can be represented as compact traces, which can then be used for scheduling and power/performance estimation through trace-based simulation. Without having to implement functional or write HDL code on an FPGA, designers are free to experiment with a wide variety of different architectures.

The scheduled graph is then translated into a compact representation format suited to its interpretation by the hardware module corresponding to the accelerator datapath model and implemented in the FPGA. The basic strategy is to interpret the DDDG, which has already scheduled both instruction and data memory operations correspond-

ing to each node of the graph. A graph called Scheduled Graph Trace (SGT) is derived from the DDDG. It has two components: one corresponding to the instruction called SGT code, and one corresponding to the data called SGT data. The SGT code and data can be run statically on the FPGA to simulate the original application. The SGT code keeps the scheduled order of the DDDG without retaining the addresses of any instructions. Every SGT operation is either an ALU instruction or a load/store instruction. Continuous data addresses are recorded inside the SGT code itself. This occurs when successive addresses vary from one another by a constant value. The SGT data is stored in a FIFO queue in the same order as the SGT codes that utilizes it. When the SGT code is executed, the next item needed is at the top of the FIFO and can be easily popped.

	Format	Description
SGT code	7 bits: Operation code 1 bit : Is new cycle	SGT format preserves the execution order of the scheduled DDDG.
Address	32 bits	Data addresses of Load/Store instructions

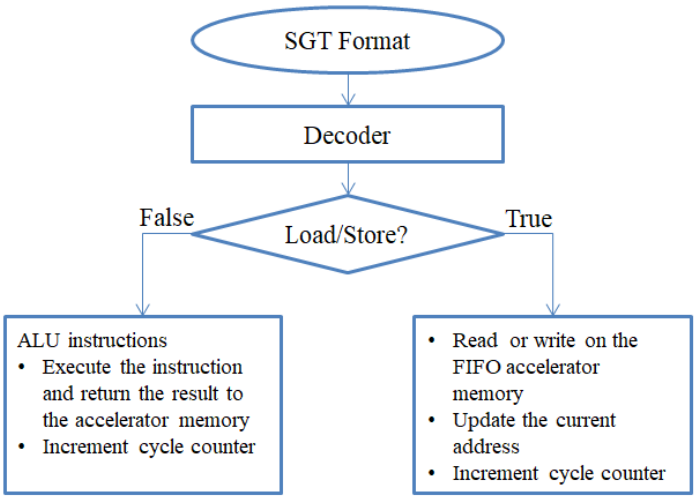


Figure 3.2 – Actions taken by the timing accelerator for the SGT format.

Figure 3.2 presents an illustration of the structure of the SGT format. It also demonstrates how the SGT format is decoded and executed by the timing component of the accelerator. The SGT code has seven bits to encode the operations of the RISC-V ISA, which can be decoded on the RISC-V processor model. SGT format preserves the execution order of scheduled DDDG with the bit "is new cycle" that is used to calculate

the performance of the accelerator. The graph is translated into three instruction categories: ALU instructions (i.e., register to register operations), load/store instructions with their data addresses, and other instructions that do not constitute loops. The ALU instructions execute the instruction and return the result to the SGT data file (accelerator memory). The instructions for load/store read or write, and update accelerator memory After each instruction, the timing model will use the *isNewCycle* bit to update the cycle counter.

3.3.3 Flow explanation by an example

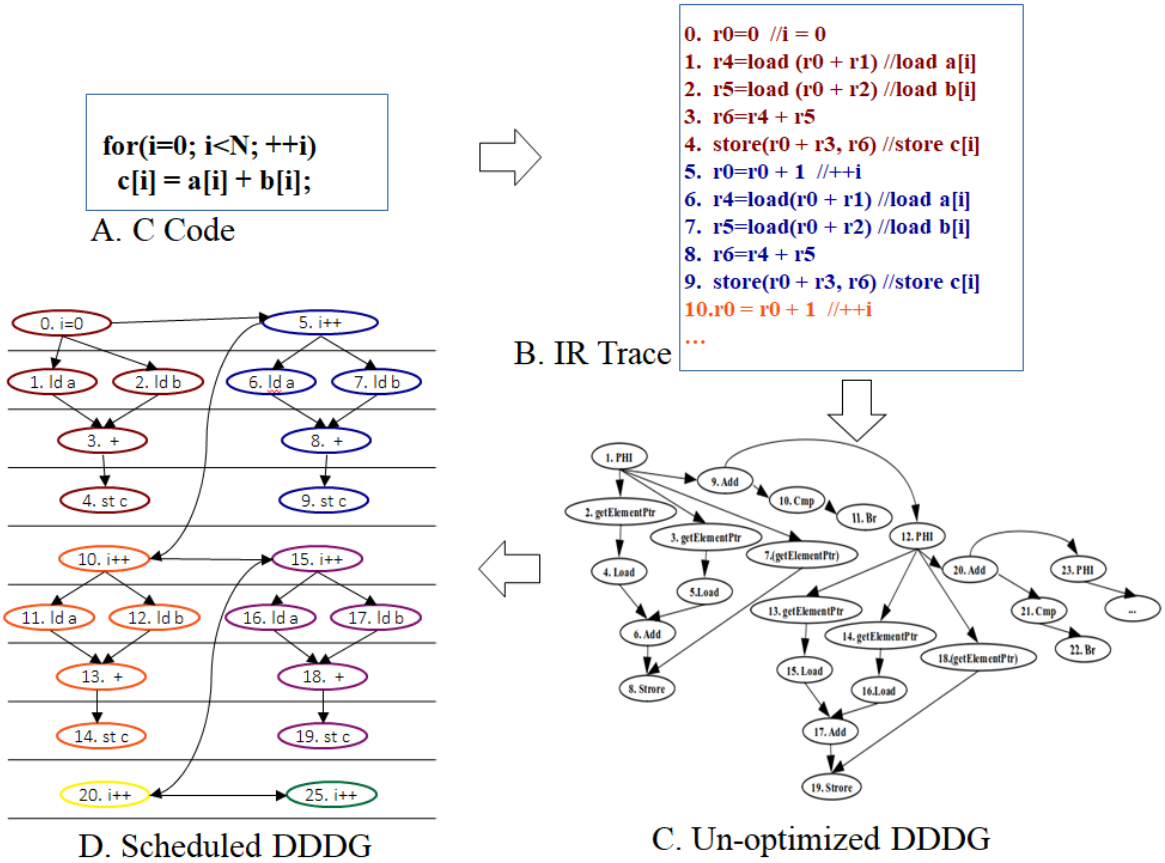


Figure 3.3 – An example of an accelerator model with a factor of 2 loop iteration parallelism, partitioning factor 2, and without loop pipelining.

Figure 3.3 illustrates the flow using an example corresponding to the addition of two vectors. The accelerator modeling steps initiate with a C description of the algorithm

and the instrumentation step that generates the application execution trace. There are instances of instruction IDs, operators, operands, and virtual register IDs in the IR trace (Figure 3.3B). In the next step, an unoptimized DDDG (Figure 3.3C) is generated as a directed and acyclic graph, with nodes representing instructions and edges representing dependencies. To schedule DDDG, let us suppose that the *add* unit operation, the memory *load* operation, and the *store* operation each have a delay of one cycle. We will assume that the designer made the decision to implement a cyclic partitioning of the array *c* with a factor of 2 together with a loop unrolling with a factor of 2 and without loop pipelining.

Optimizations are then applied to the DDDG. The optimized DDDG, shown in Figure 3.3D, reflects the dataflow nature of the accelerator. Edges in the DDDG represent flow dependencies, and DDDG node latencies are added to edges as edge weights. The scheduled DDDG is generated by the breadth-first traversal algorithm. Each memory partition has two *read* ports, memory *load* operations for vector *a* and *b* can be executed in the same cycle. Vector *c* has cyclic partitioning with a factor of 2, and two memory *stores* accessing different memory banks can be executed in the same cycle. Figure 3.3D represents a feasible schedule for which the loop iteration takes eight cycles. Finally, the scheduled DDDG is generated. The basic strategy is to remove all possible redundancies from the program execution trace while preserving fidelity.

3.4 Integration of an accelerator model in the HAsim simulator

In the previous section, we used a set of supporting tools to model specialized hardware accelerators. We also presented the processor models targeting the FPGA-based simulator (HAsim) in chapter 2 (2.3). In this section, we will show how to use HAsim to simulate heterogeneous systems by combining the accelerator and processor models. Specifically, we will focus on the Rocket Custom Co-processor (RoCC) [1] interface offered by the Rocket Core [2], the hardware blocks can be included. This interface enables the processor to connect with the hardware accelerator by executing RISC-V ISA-supported custom instructions. At the end, our method can simulate cus-

1. https://inst.eecs.berkeley.edu/ cs250/sp17/disc/lab2-disc.pdf
2. https://github.com/chipsalliance/rocket-chip

tom hardware accelerators and processors targeted at the FPGA platform.

Programming flow

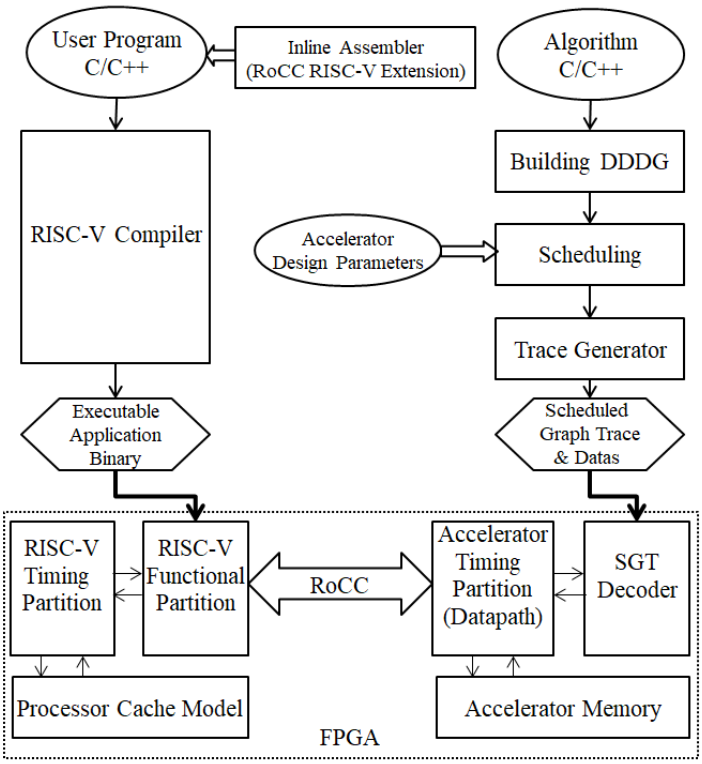


Figure 3.4 – The flow designed to generate a customized accelerator and its simulation models.

As shown in Figure 3.4, a flow has been designed to produce a customized accelerator and its simulation models. On the right side of Figure 3.4 is the process flow for generating customized accelerator models. The inputs required by the simulation model generator are the high-level C/C++ source code and design parameters for the accelerator. The C code will then be generated into a scheduled Graph Trace format, which will be decoded and interpreted by the accelerator timing partition. The accelerator timing partition (or timing model) is in charge of tracking architectural performance and communicating with the processor and its memory. The user program is initially written in C/C++, and the RISC-V compiler will then generate code compatible with RISC-V processor models. Because the RoCC custom instruction is not supported by the RISC-V compiler, it is inlined as assembly. The RoCC interface enables the RISC-V

processor to communicate with the accelerator by executing custom instructions supported by the RISC-V ISA. The compiler will generate the application binary that the FPGA's functionality will execute.

Accelerator-Processor model integration

Integration of the hardware blocks can be done through an interface similar to the rocket custom co-processor (RoCC) that is provided by the rocket core. This interface allows the processor to communicate with the hardware block by executing custom instructions supported by the RISC-V ISA. The Rocket core is an in order, single-issue, scalar processor with a 5-stage pipeline. It executes the 32/64-bit RISC-V ISA and features an integer ALU and an optional FPU. The default configuration of this core includes first level instruction and data caches. It additionally provides an accelerator or co-processor interface called RoCC. Through the interface provided by the rocket core's rocket custom co-processor (RoCC), the hardware blocks may be integrated. This interface enables the core to connect with the hardware block by executing RISC-V ISA-supported custom instructions.

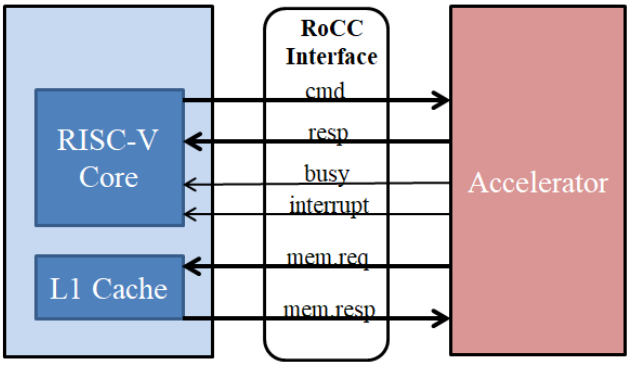


Figure 3.5 – An overview of the communications, including the rocket core and accelerator.

Figure 3.5 shows the processing tile combining the rocket core, the accelerator, and its RoCC interface. The Rocket core transmits the custom instructions together with the instruction's source registers to the RoCC interface. Consequently, the custom instructions must be executed in order to trigger the accelerator.

Figure 3.6 gives an overview of the custom instruction format. It provides two source registers and one destination register value that can be passed to the accelerator;

31 – 25	24 – 20	19 – 15	14	13	12	11 – 7	6 – 0
funct7	rs2	rs1	xd	xs1	xs2	rd	opcode
7	5	5	1	1	1	5	7
roccinst[6:0]	src2	src1				dest	*custom-0/1/2/3*

Figure 3.6 – Custom instruction format.

a function code is used to trigger a specific accelerator; additional bit fields in the instructions indicate if the processor requires an answer from the accelerator. Due to a lack of compiler support, the custom instruction needs to be inlined as assembly. The number of integrated hardware blocks might change based on the requirements of the target application.

Simulation targeting the HAsim simulator

Figure 3.7 presents the structure of the proposed HAsim-based simulation platform. The application code and structural design parameters are entered by users. Between the user and the FPGAs is a software layer running on the host CPU. It comprises a set of tools that includes Aladdin, gem5, the trace generation tool, and other software services.

The simulator uses the Aladdin flow to generate and schedule the DDDG graph. It then generates compact STG data and SGT code using the accelerator trace tool. The accelerator trace-based modeling should be interpreted on the FPGA and integrated into the processor core. Each accelerator model is composed of the accelerator timing partition with its SGT data, SGT code, and an interface . The SGT code and data are interpreted by the timing partition. The timing model of accelerator (written in Bluespec System Verilog) interfaces with the processor simulation models (functional partition, timing partition, memory system, and router). The Bluespec System Compiler is used to synthesize the simulation models and generate the FPGA bitstream. The simulator finally configures the FPGA and loads it with the SGT code, SGT data, and the RISC-V binary.

More details about the processor models and HAsim framework were given in 2.2.1. The functional partition handles correct ISA level execution of the instruction stream. The timing partition (or timing model) is responsible for tracking micro-architectural specific timings. For rare but challenging implementation events, such as system calls,

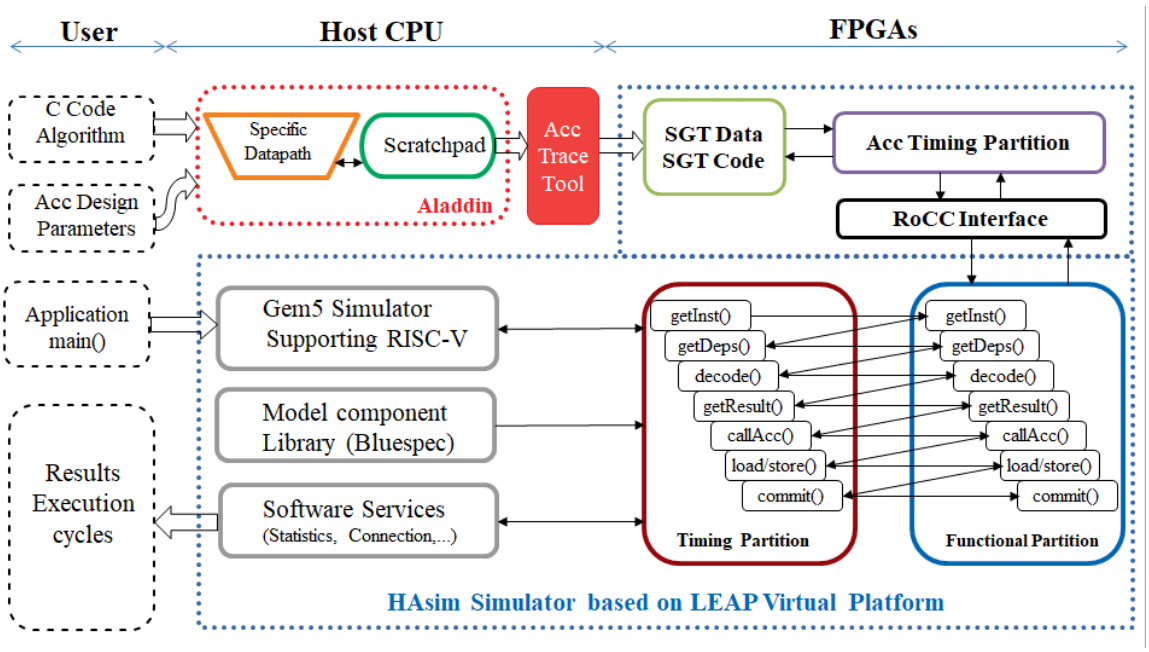


Figure 3.7 – Structure of the proposed simulation platform.

HAsim includes the gem5 simulator running on the host CPU. Additionally, to achieve flexibility and to reduce the development effort when designers can change the simulation architecture, a library of predefined modeling components allows architects to adapt pre-existing modules to their experiments, such as caches, processor models, network models, etc. The software services refer to all functionality not directly related to simulation (including the ability to track statistics and parameters, as well as the virtual platform necessary to interact with the host CPU). Finally, the simulator returns the results back to the host CPU and presents them to the user.

3.5 Performance assessment

In this section, we present a case study of the system level design and evaluation for heterogeneous SoC architectures using the FPGA-based simulator to evaluate the correctness of the design flow. The architectural models have been compiled for the Bluesim simulator, as illustrated in section 2.2.3 of chapter 2. We examine two types of experiments:

— First, we illustrate how to design a hardware specific accelerator for the Blocked

General Matrix Multiply algorithm [63]. Our results are compared to those obtained with the Vivado High-Level Synthesis (HLS) tool. The Xilinx Vivado HLS version 2018.3 is used, and the accelerator clock frequency is set to 100 MHz.

— Second, we expand this case study (in simulation only) to seven benchmarks of the MachSuite benchmark suite [98] and compare the performance of the standalone processor to different heterogeneous architectures.

3.5.1 Case study: Blocked Matrix Multiply accelerator

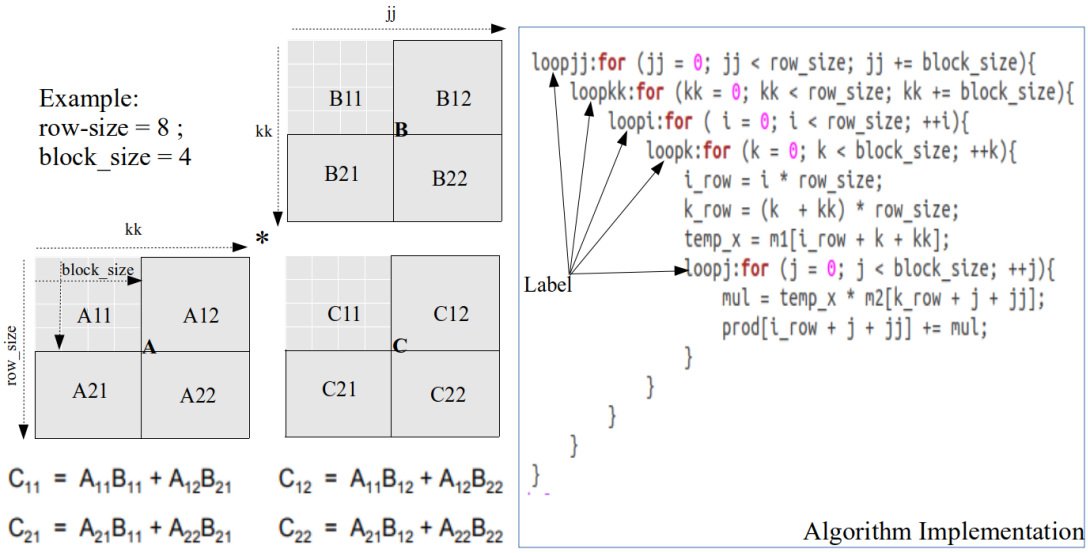


Figure 3.8 – The pseudocode of the blocked GEMM algorithm.

General Matrix Multiply (GEMM) is a common algorithm in linear algebra, machine learning, statistics, and many other domains. It provides a trade-off between space and time, as there are many ways to partition the computation. Matrix multiply is more commonly computed using a blocked loop structure. Memory locality is dramatically improved by commuting the arithmetic to reuse all of the elements in one block before moving on to the next. Implementation uses a fixed blocking factor of 4 and is based on the algorithm proposed in [63]. Figure 3.8 illustrates the core computation of blocked GEMM.

Starting from the algorithm, the accelerator model is generated, its simulation is then performed with HAsim. Vivado HLS is used as a reference tool for evaluation. A C

code application program is also developed which includes the following steps: loading input data, invoking accelerator, waiting for accelerator response, and storing results. The main program is compiled to RISC-V binary and loaded into the processor model.

As shown in Figure 3.9A, the cycle count for blocked GEMM application while considering loop unrolling, pipelining and array partitioning pragmas can be predicted. The x-axis denotes the loop unrolling factor ranging from 1 to 16.

For each configuration, we set the loop unrolling factor and apply pipelining to all loops in the algorithm (the array partitioning is not considered in this experiment). This evaluation highlighted a decrease in the number of cycles when the unrolling factor increased. In addition, the number of cycles predicted by the accelerator simulator corresponds very closely to that provided by Vivado HLS.

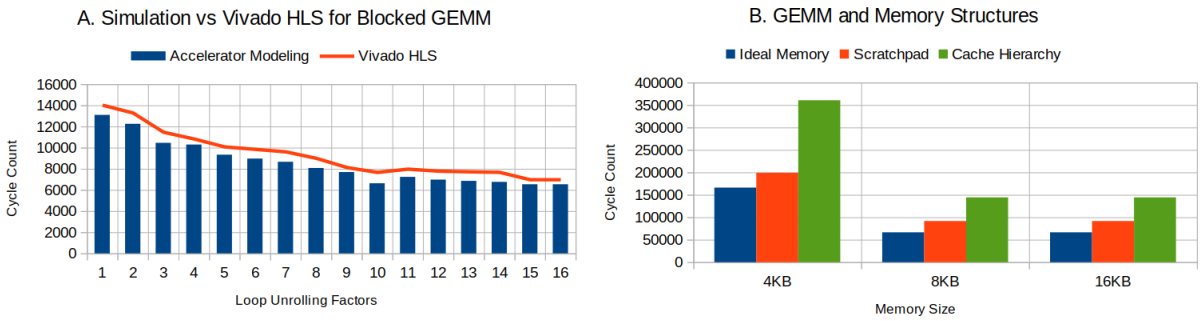


Figure 3.9 – Blocked GEMM evaluation.

Figure 3.9B compares the performance of an accelerator for the blocked GEMM benchmark according to the usage of different memory structures: one cycle memory latency (ideal memory), scratchpad memory having a fixed latency for each request, and cache model. In order to decompose the execution time, we performed simulations with the heterogeneous simulation model (processor, accelerator, cache, scratchpad, and router). This highlights that memory access time takes up a significant portion of the execution time as the complexity of memory increases. Moreover, the number of cycles decreases as the memory size grows from 4 KB to 8 KB. This reduction is explained by the fact that a 4 KB memory size is too small to store the blocked data size (a 16*16 matrix).

3.5.2 Machsuite benchmarks

We considered seven benchmarks from different domains and studied their implementation with our FPGA-based simulator. Benchmarks are chosen from Machsuite [98], with some modifications to fit the gcc RISC-V compiler. Figure 3.10 shows the estimated performance of these algorithms for different instantiations of the processor-accelerator pair. For all the applications, configurations for loop unrolling factors, pipelining options, and array partitioning factors/types are identical.

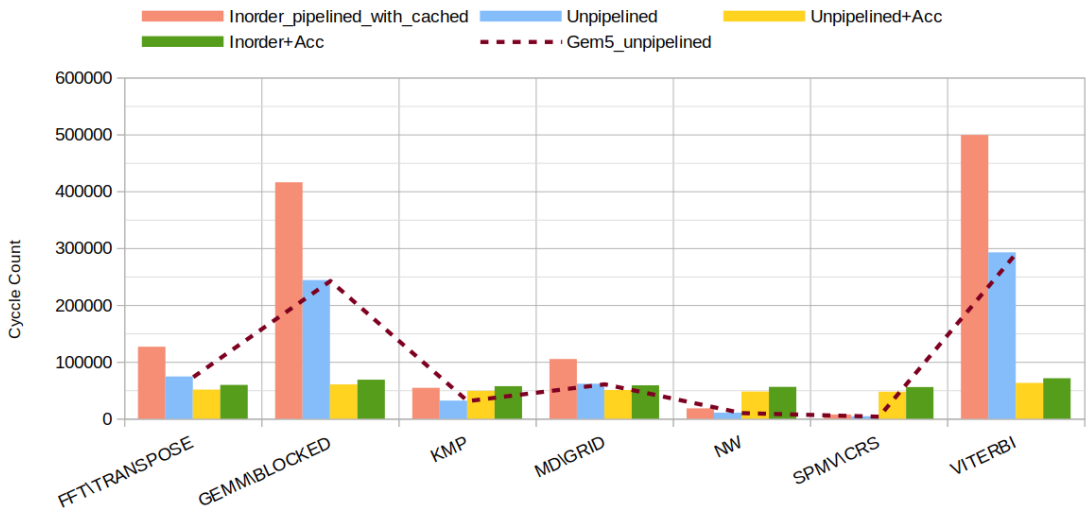


Figure 3.10 – Performance validation under different architectures.

To assess the performance of a system combining a processor core and an accelerator models, we considered that the core executes a program that make a call to the accelerator that implements each of these benchmarks.

It is not surprising to see that for every application the in-order pipelined with cache is the slowest architecture because it simulates the complex behavior of the architecture. We also see improved performance of heterogeneous models over processor-only models. The key motivation behind the proposed design method is to rapidly evaluate the processor/accelerator performance in an architecture exploration context.

Finally, we evaluated the simulator accuracy of our framework. As shown in Figure 3.10, the line shows the simulation results for the baseline gem5 simulator (unpipelined model), which is measured in number of simulated execution cycles. Experiments show that our simulator has the same level of precision as the gem5 simulator.

3.6 Conclusion

To overcome the power and utilization walls in today's SoC design, computer architects replace redundant processors with accelerators that can achieve orders-of-magnitude performance and energy gains. In this chapter, we introduced an approach for designing application-specific heterogeneous architecture based on performance models through integrating an accelerator into a RISC-V core model. The primary purpose of generating these models is to use them to simulate future heterogeneous multi-core/multi-accelerator architectures. The design approach aims to explore architectures based on the FPGA-based simulator. We developed a tool to automatically generate the data flow graph and the scheduled trace of an accelerator from applications written in C. We integrated the accelerator model into a module that executes the custom RISC-V instruction set. By using HAsim framework, we modeled an unpipelined and in-order-pipelined RISC-V processor's interfacing accelerator and memory system. The blocked GEMM case study showed that accelerators can be produced from the application program without investing any extra effort into developing the hardware. The architecture exploration is confirmed by running the MatchSuite benchmark in different heterogeneous architectures.

CHAPTER 4

DETERMINING OPTIMAL CONFIGURATION ARCHITECTURE FOR HETEROGENEOUS-ACCELERATOR SOCS

Modern SoC systems consist of general-purpose processor cores augmented with large numbers of specialized accelerators. Building such systems requires a design flow that allows the design space to be explored at the system level with an appropriate strategy. In this chapter, we focus on a methodology allowing us to explore the design space of power-performance heterogeneous SoCs by combining an architecture simulator (gem5-Aladdin) and a hyperparameter optimization method (Hyperopt). We introduce this chapter by presenting related works and then our design approach. After that, we illustrate how the hyperparameter optimization method works, using two algorithms called TPE and adaptive-TPE as well as the Hyperopt library. Then, show how the Hyperopt-gem5-Aladdin framework and the automatic design flow enable the sweeping of various forms of parallelism with loop unrolling strategies and memory coherency interfaces. Finally, the framework is applied to the design of a convolutional neural network and a multi-context CNN accelerator. We also use the methodology to find the best architecture, including its coherency interface, for a complex SoC that is comprised of six different types of accelerated workloads. All the results are studied and analyzed by way of experiments.

Contents

4.1 Introduction 93
4.2 Design space exploration using Hyperparameter Optimization . . 95
4.2.1 Synthetic view of a Heterogeneous-Accelerator SoC 95
4.2.2 Design space exploration via Hyperparameter Optimization . . 96
4.2.3 Hyperopt: Hyperparameter Optimization python library 98
4.3 Design methodology 104
4.3.1 Hyperopt-gem5-Aladdin framework 105
4.3.2 Parallel accelerator exploration 107
4.3.3 Memory coherency models exploration 108
4.3.4 Automatic architectural optimization design flow 109
4.4 Experiments 110
4.4.1 Convolutional Neural Network accelerator in a SoC 111
4.4.2 Multi-context accelerator 113
4.4.3 Coherency interface choice study 116
4.4.4 Hyperopt convergence study 118
4.5 Conclusion 120

4.1 Introduction

The energy efficiency gap between application-specific integrated circuits (ASICs) and general-purpose processors motivates the design of heterogeneous accelerator system-on-chip (SoC) architectures, the latter of which have received increasing interest in recent years [107]. To support several heavy demanding workloads simultaneously and reduce unpowered silicon area, the trend in computer architecture design is to implement many special-purpose on-chip accelerators in ASIC and share them among multiple processor cores. Such architectures offer much better performance and lower power compared to performing the same task on a general-purpose CPU. Designing heterogeneous-accelerator SoCs is extremely expensive and time-consuming. The designer has to face many design issues, such as the choice of the parallelism degree and the resource utilization of accelerators, their interfaces with the memory hierarchy, etc. Design space exploration methodologies are of major importance.

Most of the time, engineers who want to tune a design will try out some of the options to see how it responds. They will start to understand how the different choices affect the model in their minds. However, it is a difficult task to explore a huge design space without the aid of an automated system. An inexperienced designer will probably struggle to model this complex process when the response surface is complex, such as non-linear, non-convex, discontinuous, or multi-modal, ultimately missing the chance to produce high-quality hardware. Also, models of computer architecture aren't accurate because they don't take into account how things interact with each other in a complex way. This makes it hard to predict how well a computer will perform and makes it hard to design an architecture. To meet the growing demand for automatic and accurate design of heterogeneous SoC, there is a way to simulate and optimize design space in the computer systems domain. We can consider the simulation an unknown objective function that is intended to be optimized. This problem is also known as black-box optimization [43] and, in the computer systems world, as design space exploration. Approaches based on hyperparameter optimization prove to be very useful in optimizing unknown objective functions, as stated in the works presented in [104, 12]. They are more powerful than heuristic optimization in terms of convergence and quality of obtained solutions.

In this chapter, we present a methodology for designing modern SoC architectures,

which combine many specialized hardware accelerators and processor cores. We explore the design space of power/performance accelerator-based systems with a SoC simulator and determine the optimal configuration using a hyperparameter optimization algorithm. The proposed simulation infrastructure is based on the use of two tools: gem5-Aladdin [106] and Hyperopt [8]. gem5-Aladdin is an architectural simulator that supports the modeling of complex systems made up of heterogeneous accelerators. Hyperopt is a library implementing different hyperparameter optimization algorithms suitable for solving optimization problems with an unknown objective function [104], such as architecture simulation in our case. The main contributions of this work are as follows:

— A new framework for determining, at the system-level, the microarchitecture with the best efficiency, in terms of performance-power ratio.

— A case study allowing to identify the most energy efficient architecture for a convolutional neural network (CNN). We showed that the obtained solution achieves a 2x to 4x improvement in energy-delay-product compared to an architecture without parallelism. Furthermore, this solution is more efficient than commonly implemented architectures (systolic, 2D-mapping, and tiling).

— Using the framework, we proposed an accelerator for three representative CNNs and chose the configuration with the most efficient EDP. The improvement of the selected CNN accelerator configuration relative to the most optimal architectural configuration for each individual CNN ranges between 10 and 13.4 %.

— To demonstrate the efficiency of the heterogeneous-accelerator SoC design approach, we determined the optimal architecture, including its coherency interface, for a complex SoC made up of six common accelerated-workloads. Three possible coherency models are considered: a software-managed direct memory access (DMA), a shared last level cache (LLC-coherent), and a fully-coherent cache. Our framework has shown that a hybrid interface appears to be the most efficient, with a 22 % and 12 % improvement in EDP compared to just only using non-coherent and only LLC-coherent models, respectively.

4.2 Design space exploration using Hyperparameter Optimization

This section describes optimization techniques based on hyperparameters that have been demonstrated to be highly effective for optimizing unknown objective functions. These methods are better than heuristic optimization in terms of convergence and the quality of the obtained solutions. This section is organized the following way: First, we present our view of heterogeneous architecture, which is targeted in this thesis. Then, Bayesian hyperparameter optimization (also referred to as sequential model-based optimization, SMBO) is introduced. Finally, we present the *Hypertopt* Python library, which is used to construct a probability model of the objective function and select the most promising hyperparameters for evaluating the true objective function. The TPE and adaptive-TPE algorithms, which are the heart of the method as well as an example of how the Hypertopt works, are both explained and demonstrated here.

4.2.1 Synthetic view of a Heterogeneous-Accelerator SoC

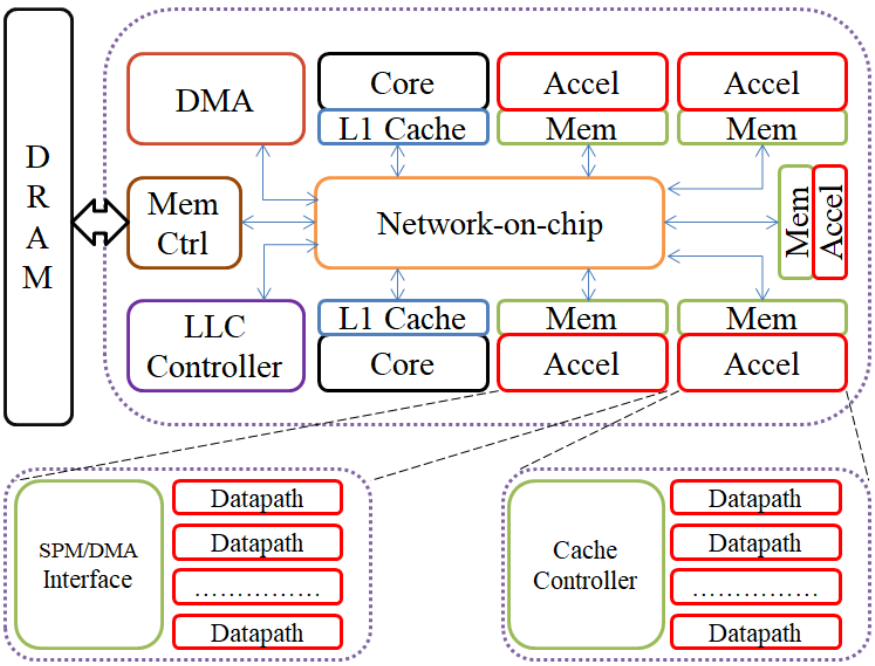


Figure 4.1 – View of a typical heterogeneous-accelerator SoC.

Figure 4.1 depicts a view of a typical heterogeneous-accelerator SoC, which consists of general-purpose cores, memory controllers, a DMA engine, and several types of specialized accelerators, all of which are connected through a configurable network-on-chip (NoC). Each accelerator is constructed out of a number of dedicated datapaths, each of which implements a part of an algorithm or the entire algorithm for a specific application domain. The accelerator has scratch-pad memory (SPM) or a cache controller for its local memory. Each accelerator uses a software-programmed SPM and a direct memory access (DMA) engine in order to achieve high performance. These two components allow for communication with the other components of the cache memory hierarchy. Additionally, there are last-level cache (LLC) and memory controllers that are coherent and shared by both the processor cores and the accelerators. This is done in order to provide high bandwidth to the accelerators.

4.2.2 Design space exploration via Hyperparameter Optimization

Design-space exploration (DSE) is usually needed in the design process to determine the best choices for the target architecture. Simulators are often used to perform DSE in the early stages of computer architecture design [29, 73]. At the system level, the DSE is inherently a component-based design effort since the choice of a RTL implementation of a module must be made in the context of all the other modules that are also components. In the multi-objective design space of the whole SoC, a set of decisions leads to a specific point. As a result, the method is used to obtain the diagram of Pareto-optimal points and is repeated hierarchically at the system level [71].

Recent computer systems and architectures have been optimized for machine learning (ML) models. System and computer architecture may be improved with the assistance of ML [121]. A model of machine learning is a formula with several parameters that must be learned from data. These parameters correspond to "higher-level" characteristics of the model, such as its complexity or learning rate. They are known as hyperparameters. This might include, for example, the number of trees in a random forest, the number of hidden layers in neural networks, the design of a telecom network, etc.

Automated hyperparameter optimization can reduce human effort, improve productivity, and be more fair in scientific studies [44]. Approaches based on hyperparameter optimization prove to be very effective in optimizing unknown objective functions, as

stated in the works presented in [104, 12]. They outperform heuristic optimization in terms of convergence and quality of solutions. We refer to the computer architecture simulation as a "black-box function" which lacks an algebraic system model to be optimized. We can also assess the potential of space for configurations for heterogeneous SoC design because hyperparameters must be quickly optimized in order to obtain an architectural solution. Our work focuses on developing a methodology that relies on hyperparameter optimization to help select an optimal architecture.

There are two primary approaches: the random search and the grid search. If each training session takes several hours or even days to complete, it is unrealistic to conduct such an exhaustive search because it would be extremely expensive. It would be impossible to try all of the cases that are contained within the search space, so it is strongly recommended to exploit the search result. For instance, after we have tried a few different points in the search space, we will have a better idea of which areas of such a space produce good results and which areas do not produce good results. Therefore, we are going to be focused on a concept that is known as SMBO, which stands for sequential model-based optimization [56].

```
SMBO(f, M_0, T, S)
    H ← ∅,
    For t ← 1 to T,
        x* ← argmin_x S(x, M_{t-1}),
        Evaluate f(x*),          ▷ Expensive step
        H ← H ∪ (x*, f(x*)),
        Fit a new model M_t to H.
    return H
```

Figure 4.2 – The pseudo-code of generic Sequential Model-Based Optimization [9].

The pseudocode of a generic SMBO is shown in Figure 4.2. The method starts by evaluating a function *f* called true function and establishing the search history $\mathcal{H}$. The following procedure steps are done iteratively *t* times, where T is the budget for the number of iterations. SMBO algorithms choose promising configurations x^* from a model *M* that models f based on the history of observations $\mathcal{H}$ at each iteration. There are two innovative techniques *S* for estimating f by modeling $\mathcal{H}$ [9]: a hierarchical Gaussian Process and a Tree-structured Parzen Estimator (TPE).

Bayesian optimization is a machine-learning-based method for the global optimization of black-box functions that is widely applied to many real-world problems [109, 110, 32, 80]. It implements a probabilistic surrogate model and an acquisition function iteratively. The probabilistic surrogate models the unknown objective function based on already observed samples (search history). The acquisition function optimizes the surrogate model to decide which point to evaluate next.

Bayesian optimization can figure out how the parameters work together and ignore configurations that have a low chance of being good solutions, so that the search can be done more globally. Thus, it can speed up the search and better avoid bad local if the current state is on a plateau or a bad local minima. So, Bayesian optimization seems like a good way to speed up exploration of the design space for computer architecture. Based on these features and properties, we propose a framework that applies Bayesian optimization to an architecture simulation in order to make early design exploration more efficient. Bayesian optimization methods, such as gaussian processes (GP), Random Forest, and Tree-structured Parzen Estimator, use different ways to build surrogate models (TPE). Tree-based models like Random Forest and TPE are appropriate for solving large-size discrete domain problems [39]. GP is often used for moderate-size problems with a continuous domain. In our work, we chose TPE for Bayesian optimization because it is well suited for discrete problems, and architecture configuration is used to explore design.

4.2.3 Hyperopt: Hyperparameter Optimization python library

We rely on the use of a hyper-parameter optimization library called *Hyperopt* [11] to choose the design that is the most efficient. *Hyperopt* offers an optimization methodology that distinguishes between a configuration space and an architectural simulation function that associates power/performance values to configuration space locations. When using *Hyperopt*, the SMBO algorithm may be replaced with any other search algorithm. In the library, there are currently three methods [10]: random search, TPE, and the adaptive TPE algorithms. The adaptive TPE (ATPE), which is an extension of TPE, is a model that uses machine learning to automatically tune the hyperparameters.

Tree-structured Parzen Estimator (TPE algorithm)

The SMBO algorithm narrows the search space sequentially based on previous results and is used for hyperparameter optimization tasks with high dimensions and small evaluation budgets. The tree parzen estimator (TPE) is a model and Expected Improvement (EI) the optimization strategy for the SMBO algorithm. It has been thoroughly validated by a large number of people who have used it in a variety of machine learning algorithms and demonstrated its usefulness. As a result, we chose to employ it as a strategy in our research. The goal of the TPE algorithm is to maximize (EI). EI is given by equation 4.1, where variables and their probabilities are x, $P(x)$ for hyperparameters, y, $P(y)$ for the objective function, and $y*$ as the best value found after observing history $\mathcal{H}$.

$$EI_{y*}(x) = \int_{-\infty}^{\infty} max(y * -y, 0) \frac{P(x \mid y)P(y)}{P(x)} \, dy \tag{4.1}$$

$P(x \mid y)$, which is the probability of the hyperparameters given the score of the objective function, is expressed as:

$$P(x \mid y) = \begin{cases} l(x) & \text{if } y < y* \\ g(x) & \text{if } y \geq y* \end{cases} \tag{4.2}$$

TPE models the density of a parameter for "good" results and compares it to its density for "bad" results. It can then use these models to determine the expected improvement of the objective function for any value a parameter can take. The TPE algorithm depends on a value called $y*$, which is either the desired result or the best value found by looking at the past. The density that is formed by using the observation parameter x in such a way that the result of function y is less than $y*$ is denoted by $l(x)$ (good results), while the density that is formed by using the remaining observations is denoted by $g(x)$ (bad results). Once $l(x)$ and $g(x)$ have been expressed, TPE is able to identify the next parameter to consider using an equation (4.4).

$Gamma$ (γ) represents the quantile of the search result. Lower values of γ correspond to bad results, and higher values of γ correspond to good results. On The TPE method by γ is defined by the following equation:

$$\gamma = P(y < y*) = \int_{-\infty}^{y*} P(y)\, dy \tag{4.3}$$

TPE picks the previous value $y*$ to be some quantile of the observed current y values, but there is no need for a specific model for $P(y)$. By keeping sorted lists of observed variables in a search history, the runtime of each iteration of the TPE method can scale linearly with history $\mathcal{H}$ and the number of parameters being optimized.

$$EI_{y*}(x) \propto (y + \frac{g_{(x)}}{l_{(x)}}(1-\gamma))^{-1} \tag{4.4}$$

Equation 4.4 [9] illustrates the parametrization used in the TPE algorithm to facilitate the optimization of EI. Points x are looked at with the goal of maximizing improvement if they have a high probability under $l(x)$ and a low probability under $g(x)$. The tree-structured form of l and g makes it simple to generate multiple candidates based on l and evaluate them based on $g(x)/l(x)$. As a result, the algorithm delivers the candidate $x*$ with the highest EI at each iteration.

Adaptive-TPE (ATPE) algorithm

TPE may be viewed as an algorithm that predicts the optimal trial to try next based on past trials. The only fundamental hyperparameter for TPE is gamma (γ). However, we can modify the value of some parameters to improve the effectiveness of the TPE algorithm:

— $n_startup_trials$: random sampling is used instead of the TPE algorithm until a given number of trials is achieved.

— $seed$: seed for the random number generator.

— $n_ei_candidates$: number of candidate samples used to calculate the expected improvement.

— $gamma$: a function that forms a density function for samples that have low grains. It takes the number of completed trials as an input and returns the number of trials as an output.

— $prior_weight$ the weight of the prior.

— $locking_value$: a hypothetical way that could force TPE to spend more time exploring specific hyperparameters by 'locking in' the values for the others for some trials.

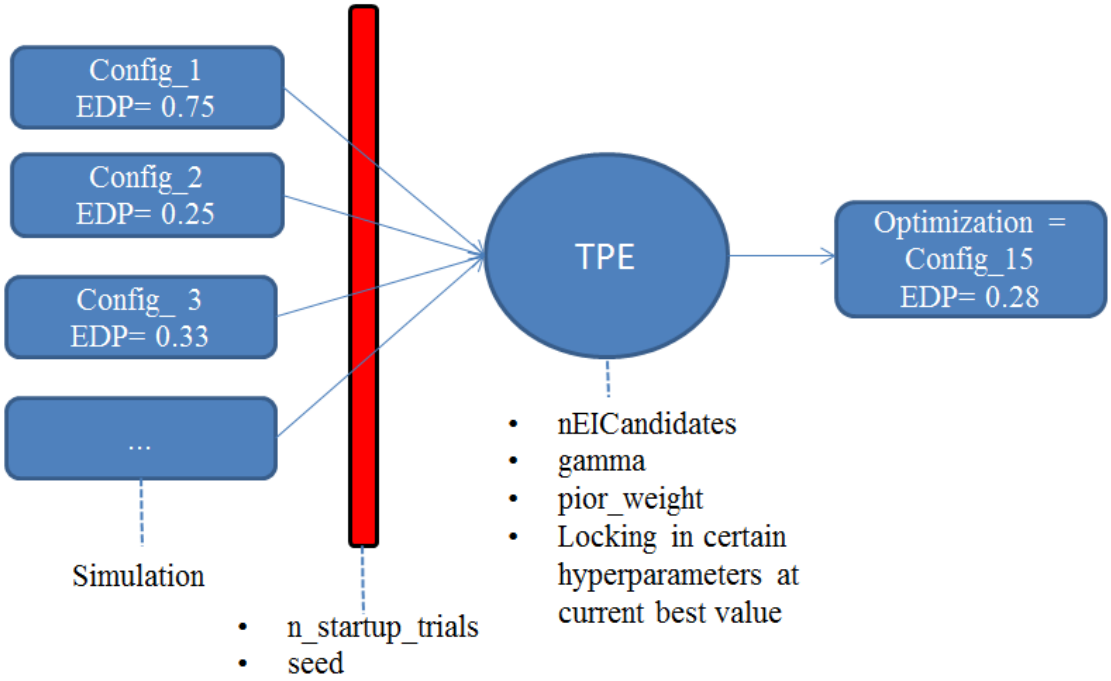


Figure 4.3 – Adaptive-TPE algorithm: a tuning parameters technique for improving TPE.

The adaptive tree of parzen estimators (ATPE) [1] is a method that enhances the behavior of the TPE algorithm. Figure 4.3 shows the parameters that can be tuned to explore more widely (more random search) or more narrowly (closer to current best)

1. https://github.com/electricbrainio/hypermax

based on the situation. In order to improve TPE [8], the authors searched out the optimal values for those parameters across a wide variety of different simulated hyperparameter spaces. They used this dataset to build a machine learning model that could predict the TPE algorithm's optimal parameters. This algorithm is called Adaptive-TPE (ATPE). ATPE is the optimization algorithm that uses a pre-trained machine-learning model to help optimizing faster and more accurately.

An example of using the Hyperopt library

The example considers an architecture where configurations result from a code consisting of six nested loops. A history table including a set of six parameters and their Energy Delay Products (EDP) is shown in Figure 4.4. After being simulated with the parameters of an architectural configuration, we got an EDP. The EDPs are sorted, saved, and used in order to partition the space into two separate sets, namely, "good results" and "bad results".

A search space is a stochastic expression that always evaluates to an argument that can be used as an input to the objective function. The hyperparameters are referred to as stochastic expressions. In this example, each group fits the configuration distribution of architecture parameters. We are able to match the configuration space utilizing categorical variables as well as uniform, log-uniform, and quantized log-uniform variables. We will refer to the top distribution as l and the bottom distribution as g.

The next step is to choose the next configuration to be evaluated. The output of the TPE method is the absolute minimum of the bottom distribution divided by the top distribution. We take this value and use it as our argument's minimum. The objective here is to maximize the good result while minimizing the bad result.

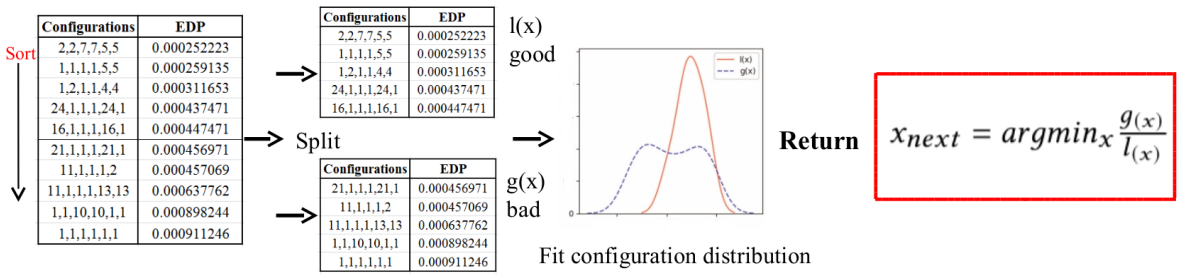


Figure 4.4 – An example of the TPE procedure with six loop iteration parallelism factors and its Energy-Delay-Product.

```
import hyperopt.pyll.stochastic
from hyperopt import hp, fmin, tpe, atpe, space_eval, STATUS_OK, STATUS_FAIL

# define a search space
configs = {
    'loop_m': hp.quniform('m', 1, 6, 1),
    'loop_n': hp.quniform('n', 1, 1.1, 1),
    'loop_r': hp.quniform('r', 1, 28, 1),
    'loop_c': hp.quniform('c', 1, 28, 1),
    'loop_i': hp.quniform('i', 1, 5, 1),
    'loop_j': hp.quniform('j', 1, 5, 1)
}

# define an objective function
def objective(configs):
    results = simulation(space)
    edp = float(result[0])
    return {'loss' :edp, 'status' : STATUS_OK}

# minimize the objective over the space
best = fmin(
        fn = objective,
        space = configs,
        algo = tpe.suggest,
        max_evals = 100)

print hyperopt.space_eval(space, best)
# -> ('1,1,17,17,5,5', 0.0342061536}
```

Search space

Objective function

Run optimization

Figure 4.5 – An example of a Hyperopt configuration with six loop iteration parallelism factors.

The way the configuration search space, the objective function, and the optimization algorithm setup are illustrated in Figure 4.5. A configuration search space object defines the searchable domain within which Hyperopt is allowed to operate. In this example, the architecture configuration parameters include six values. $hp.quniform(label, low, high, q)$ is a function that identifies a line using the formula $round(uniform(low, high)/q) *q$. This function is appropriate for use with discrete values in relation to which the objective is still somewhat smooth. For instance, if we desire to search for $loopm$ across the value of m from 1 to 6, each value is incremented by 1. This configuration also applies to loop n, r, c, i, and j.

An objective function is a python function that accepts a single argument for x (which may be any object) and returns the $f(x)$ incurred as a result of that input. Although Hyperopt supports objective functions with more complicated parameters and return values, we will utilize the simulator as an objective function that receives an array of architecture configurations and returns an EDP. This indicates that the objective function will initiate the architectural simulation and return power/performance results that correspond to the configuration.

In order to carry out optimization, we first call the $fmin()$ function, which allows us to search through the available options for the best possible objective function. In order to select a search algorithm, it is essential to assign the $algo$ keyword parameter to $hyperopt.tpe.suggest$. This will allow you to narrow down your options. Random search, also known as $hyperopt.rand.suggest$, and TPE search, also known as $hyperopt.tpe.suggest$, are two of the search algorithms that are now supported. There is also an extended version of the ATPE search method available.

4.3 Design methodology

The purpose of this section is to introduce Hyperopt-gem5-Aladdin, a framework that automatically determines the optimal architecture by simulating the dynamic interactions between accelerators and the SoC platform. We describe an automatic design flow that makes use of this framework.

4.3.1 Hyperopt-gem5-Aladdin framework

Choosing the optimal architecture and its coherency interface for each accelerator in a complex SoC including several heterogeneous accelerators is a difficult and time consuming task. Each accelerator's performance is impacted by a number of factors, including its own computation time and memory access patterns, as well as competition with other accelerators for shared resources. We propose a solution to automatically select the optimized accelerator architecture and the coherency interface for each accelerator in a heterogeneous SoC architecture. Figure 4.6 illustrates the interaction between the tools gem5-Aladdin simulator and Hyperopt involved in our framework.

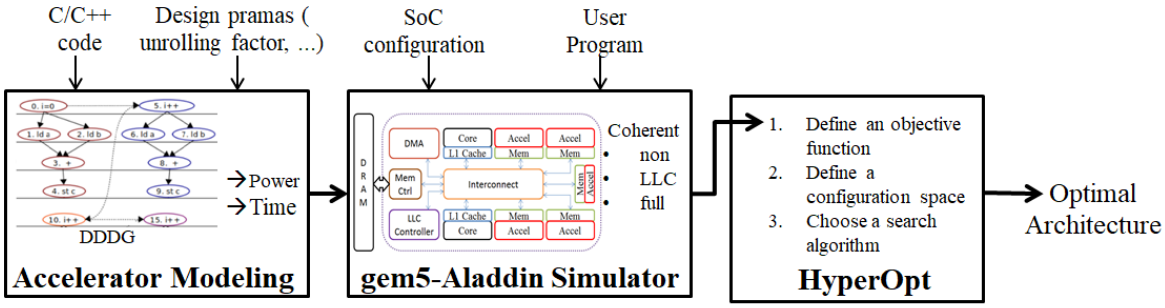


Figure 4.6 – Overview of our generic design flow using the hyperparameter optimization-based method.

The accelerator modeling is based on Aladdin [105] whose target model is the standalone datapath and its local memories. The Aladdin trace-based accelerator simulator profiles the dynamic execution trace of an accelerated workload initially expressed in C and estimates its performance, power, and area. The process mainly consists in building a dynamic data dependence graph (DDDG) of the workload which can be viewed as a data flow representation of the accelerator. This graph is then scheduled taking into account the resource constraints by the way of user-defined hardware parameters such as loop unrolling, loop pipelining, and number of memory ports. The underlying model of the Aladdin simulator is a standalone datapath and its local memories.

The interactions at the SoC view level, that is, between the accelerators and the other components of the system, are managed by gem5-Aladdin [106], which realizes the coupling of Aladdin with the gem5 architectural simulator [14]. gem5-Aladdin is able to evaluate interactions between accelerators and processor cores, DMAs, caches, and virtual memory in SoC architectures, as illustrated in Figure 4.6. With some modifica-

tions to gem5, a user program running on a CPU can invoke accelerators via a system call *ioctl*. An interrupt mechanism will wake up the CPU when the accelerators finish. It helps to put the CPU in sleep mode and eliminate polling while waiting for the accelerators to finish. gem5-Aladdin supports three coherency models for accelerators: (i) non-coherent: using software-managed DMAs; (ii) LLC-coherent: by directly accessing the coherent data in the last-level-cache (LLC) without having a private cache; (iii) fully-coherent caches: each accelerator can use its private cache to access the main memory. These coherency models are described in subsection 4.3.3.

The potential parallelism of an accelerator is expanded by design pragmas such as the loop unrolling factor during the accelerator modeling phase. This phase is used to evaluate and update the power-performance of accelerators. The gem5-Aladdin simulator can model SoCs including several accelerators that can use different coherency models, and run various workloads concurrently. The complete SoC is specified using a SoC configuration file which describes the configuration of processors, accelerators, memories/caches, and interconnect. The gem5-Aladdin simulation is an objective function of the optimization method; it provides the performance, power, delay time of SoC architectures that one wishes to optimize.

Algorithm 1: TPE-based Optimization Pseudo-Code

Data: Architecture design space X, sampling size N, random initial configurations k

Result: A parameter vector x with the minimum $f(x)$

1 Randomly simulate k architecture configurations;
2 Define initial $search_history$ with k pairs $(x, f(x))$;
3 $n \leftarrow 0$;
4 **while** $n \leq (N - k)$ **do**
5 Construct models density functions $g(x), l(x)$;
6 $x_{next} = argmin_x \frac{g_{(x)}}{l_{(x)}}$;
7 Simulate x_{next};
8 Update $search_history \leftarrow (x_{next}, f(x_{next}))$;
9 $n++$;
10 Return the best $(x, f(x)_{min})$;

Using Hyperopt, the following steps are required to explore the architecture space:

— Design an objective function that performs a simulation of configuration points and gives the power/performance value.

— Define a configuration space of valid configuration points.

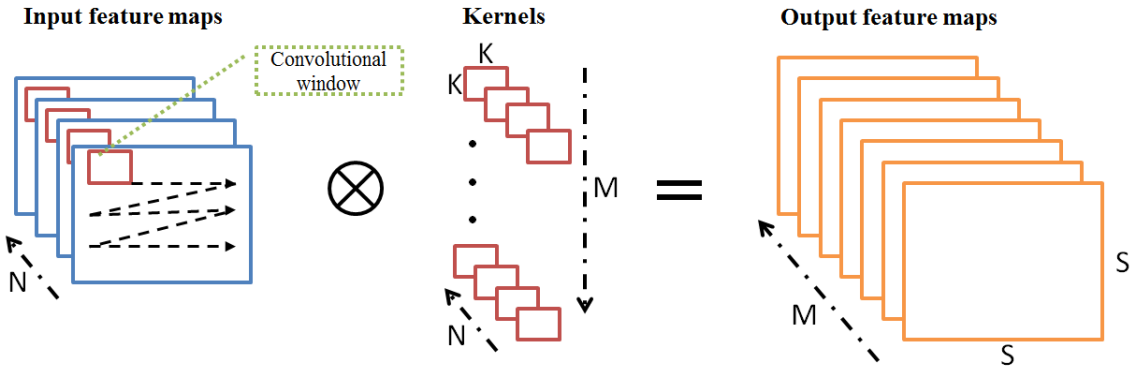
— Select a search algorithm to optimize the objective function.

Algorithm 1 presents the pseudo-code of the TPE-based method used in the framework. The algorithm starts by randomly selecting k architecture configurations and simulates them with the gem5-Aladdin simulator (line 1). The *search history* is initiated (line 2), it consists of k pairs of configurations and their associated simulation result $f(x)$. The next steps of the method are iterative and are performed N-k times, where N is the budget for the number of architecture simulations. The search space is narrowed down from the *search history* and a new configuration for the next simulation step is suggested using equations presented in subsection 4.2.3 (lines 5, 6, 7, 8, 9). Once all the iterations have been completed, the optimal architecture configuration set which reaches the minimum $f(x)$ is selected (line 10).

4.3.2 Parallel accelerator exploration

Applications in domains such as multimedia and telecommunications often spend a significant amount of time running a number of time-critical code segments with well-defined features, making them suitable for architectural specialization. These computation-intensive sections, generally loops, can be mapped onto hardware accelerators in order to increase the overall performance of the program. In addition, these computation-intensive tasks often have a high degree of inherent parallelism. Therefore, it makes sense to exploit this parallelism, and it is crucial to leverage its efficiency while developing these accelerators.

Loop nests of the considered workloads define the exploration space, as illustrated in Figure 4.7, with the convolutional layer of a Convolutional Neural Network (CNN) application. The convolutional layer (CONV) of a typical CNN application exhibits intensive parallelism at the feature map, neuron, and kernel levels. There are four parameters: M (number of output feature maps), N (number of input feature maps), S (output feature map size, or number of neurons), and K (kernel size). This is an interesting example to explore the architecture of parallel computing accelerators because it is possible to play with different loop unrolling factors with the 6 nested-loops corresponding to the CONV computation as illustrated in Figure 4.7.




```
loop_m: for(m=0; m<M; m++) {    //output feature maps loop
    loop_n: for(n=0; n<N; n++) {    //input feature maps loop
        loop_r: for(r=0; r<S; r++) {    //ouput neuron rows in a feature map
            loop_c: for(c=0; c<S; c++) {    //output neuron colums in a feature map
                loop_i: for(i=0; i<K; i++) {    //kernel rows operations
                    loop_j: for(j=0; j<S; j++) {    //kernel colums operations
                        //loop body : multiply-accumulate (MAC) operations
                        Output[m,r,c] += Input[n,r+i,c+j] * Kernel[m,n,i,j];
        }}}}}}
```

Figure 4.7 – Convolutional layer operation of a CNN.

4.3.3 Memory coherency models exploration

When designing an accelerator, the system view is of great importance and it is essential to take into account how the different components interact with each other. Indeed, the system must be designed to take into account the bandwidth available with the memory so that the accelerator units are correctly supplied with data. It is a question of considering data movement between hardware components and coherency management for accelerators. Giri et al. [51] identified three common coherency interfaces used to integrate accelerators with the memory hierarchy in a loosely coupled architecture.

— In a non-coherent interfacing model, the accelerator has a scratchpad memory (SPM) for local storage and uses DMA to load data from DRAM, as illustrated in Figure 4.1.

— LLC-coherent accelerators send DMA requests to the last level cache. Their implementation is similar to that of non-coherent accelerators, but the LLC-coherent DMA requests/responses are routed to the cache-coherent module instead of the DMA.

— In a fully coherent model, each accelerator has its private cache which imple-

ments a cache coherence protocol such as MESI or MOESI, similar to a processor's cache.

Each of these three coherency models offers interesting power-performance trade-offs. For instance, overheads due to the costly main memory accesses increase contention when operating concurrently. Irregular data access patterns increase LLC misses, and generate a significant hardware area/power overhead. In a SoC integrating several accelerators to support a versatile application, a single coherency interface used by all accelerators may not be the most optimal in terms of power and performance as shown by Giri et al. [51]. Each accelerator may have its coherency model, and this is what we explore with the flow.

4.3.4 Automatic architectural optimization design flow

Hardware designers often use the heuristic design method, which in the future will increase design time and expenses. Hardware design automation is motivated by the need to enable novice architects, software engineers, and algorithm developers to participate in hardware design. Consequently, an efficient design flow from algorithms to architectures and the selection of an automatically optimized solution are essential. Approaches based on TPE algorithm optimization must define the number of samples and explore them all in order to get the results. In addition, we must begin from scratch when we run optimization again. This is time-consuming when simulating a complex architecture and a huge design space, as each simulation may take a significant amount of time. To overcome this drawback, we employ a multi-step HyperOpt-gem5-Aladdin optimization that can automatically select the optimal architecture design without requiring a predetermined number of simulations. Additionally, it stores the search history, allowing the user to continue exploring the design space without having to start over. The multi-step optimization procedure is demonstrated in algorithm 2.

The designer will first define expected results, such as the desired level of performance or power value. The notations S, N, and $f(x)_{min}$ represent, respectively, the size of the design space, the number of simulations performed, and the optimal value for each step. In the first step of the procedure, the $search_history$ is defined, which includes the architecture parameter vector x along with the simulation result $f(x)_{min}$. The investigation can be saved and resumed without necessitating re-running simulations. The budget for the number of designs that the *Hyperopt-gem5-Aladdin* framework will

produce is denoted by the parameter N. Step-by-step, the design space is explored so that the *Hyperopt-gem5-Aladdin* framework is invoked and the $search_history$ is updated (lines 5, 6, 7). As a prerequisite for stopping the exploration at each successive step, the most optimal value is compared to the Expected Result (ER). Or we try all spaces (S) until we reach the point where the ER value is closest. ER is defined by the designer based on the requirements of the target architecture.

Algorithm 2: Pseudo-code For Automatic Architectural Optimization Flow

Data: Architecture design space X, design space size S, sampling size of Hyperopt-gem5-Aladdin N, the expected result is the desired outcome for the designer ER.

Result: An architecture parameter vector x with the minimum result of the simulation $f(x)_{min}$

1 Define expected result ER;
2 Define initial $search_history$;
3 $i \leftarrow 0$;
4 **while** $(f(x)_{min} - ER) > 0)$ or $((i * N) < S)$ **do**
5 $\quad (x, f(x)_{min}) \leftarrow$ *HyperOpt-gem5-Aladdin*(N);
6 $\quad$ Update $search_history$ with N pairs $(x, f(x))$;
7 $\quad i++$;
8 Return the best $(x, f(x)_{min})$;

4.4 Experiments

In this section, we investigate the design space and optimize the configuration for the Convolutional Neural Network Accelerator in a SoC utilizing our hyperOpt-gem5-aladdin framework. Through experiments, we were able to find the CNN architecture that achieves high performance while using the least amount of energy. We also looked into a CNN accelerator that can run different CNN applications in different contexts (a multi-context accelerator). Also, we figured out the best architecture for a complex SoC with six common accelerated workloads, including the coherency interface. Finally, we compared the convergence of random search, traditional TPE, and ATPE.

Our studies aim to characterize, in terms of energy delay, the system-level exploration of the architectural design space of a heterogeneous accelerator SoC. We dynamically estimate the value of the associated Energy-Delay Product (EDP) for each

point in the design space, taking into account both performance and energy limitations. The energy usage in the EDP function was indicated in Joules Per Instruction (JPI), while the execution latency was given in Clock Cycles Per Instruction (CCPI or CPI). The EDP has been chosen as the assessment measure for comparing alternative designs in terms of various parameters. Through the use of EDP, our experiments focus on memory coherency models and parallel architecture exploration.

The experimental setup for the modeled SoC includes the processor configuration and the accelerators that will be designed. Our framework is based on gem5-Aladdin, and most configurations are the same for all experiments. Table 4.1 outlines the system-on-chip configuration of the gem5-Aladdin simulator used for the experiments.

Table 4.1 – gem5-Aladdin SoC Architecture Configuration.

Component	**Description**
CPU Type	Out-of-order X86
System Clock	100MHz
Cache Line Size	64 bits
L2 Cache (LLC)	2 MB, 16-way, LRU
Memory	DDR3_1600_8x8, 4 GB
Hardware Prefetchers	Strided
Data Transfer Mechanism	DMA/Cache

4.4.1 Convolutional Neural Network accelerator in a SoC

As presented in subsection 4.3.2, Figure 4.7 depicts an example of a convolutional layer of a convolutional neural network (CNN) application. The layers of a standard CNN include convolution (CONV), pooling, and fully connected. The first two layers, convolution and pooling, extract features, while the third layer, fully connected, translates the retrieved features to the final output, such as classification. In a typical implementation, the CONV layers use more than 90% of the execution time during the inference [31]. CNN layers are highly computation intensive and exhibit fine-grained parallelism at feature map (FP), neuron (NP), and synapse (SP) levels. This potential parallelism offers many opportunities to speed up the calculations. However, most existing CONV accelerators exploit the parallelism only at one level [72]. Systolic architectures can usually exploit synapse parallelism [17], 2D-Mapping architectures neuron parallelism [37], and Tiling architectures feature map parallelism [18].

There is a lack of architectural studies trying to exploit these different types of fine-grained parallelism simultaneously. By exploring all possible types of parallelism, and depending on user constraints, greater efficiency can be expected.

The calculations of a CONV layer, as shown with the code in Figure 4.7, can be unrolled in different ways. The labels in the code ($loop_m, loop_n, loop_r, loop_c, loop_i, loop_j$) are used to set the unrolling factors and quantify the parallelism degree of each loop. According to the different unrolling strategies of the loops, there are three types of parallelism.

— Feature map Parallelism (FP), $loop_m$ output feature maps, and $loop_n$ input feature maps are processed at a time (maximum factors are M and N respectively).

— Neuron Parallelism (NP), $loop_r$, and $loop_c$ neurons of one output feature map are processed at a time (maximum factor is S).

— Synapse Parallelism (SP), $loop_i$, and $loop_j$ synapses of one kernel are computed at a time (maximum factor is K).

The design space is built by combining these three types of parallelism. As an example, an architecture may handle a single input feature map and a single output feature map ($loop_m = 1$ and $loop_n = 1$), one neuron of each output feature map ($loop_r = 1$ and $loop_c = 1$), but multiple synapses of each kernel at a time ($loop_i > 1$ or $loop_j > 1$). This corresponds to a style of parallel computing named Single Feature map, Single Neuron, Multiple Synapses (SFSNMS). It is obviously possible to define other processing styles: SFSNSS, SFMNSS, SFMNMS, MFSNSS, MFSNMS, MFMNSS and MFMNMS [72].

We evaluated three common workloads selected from published papers. LeNet-5 [66], the most famous handwriting recognition model, FR[34] implementing a face recognition model, and HG [70] used to recognize hand gestures of humans. In this experiment, we used a non-coherent interface model, it has a private scratchpad memory for local storage and uses DMA to request data from the main memory.

Table 4.2 gives the configuration of three well-known parallel architectures (tiling, 2D-mapping and systolic) for each of the considered workloads. The fourth architecture, called *selection*, corresponds to that resulting from our exploration.

Figure 4.8 shows the EDP results for the six workloads. In this figure, the horizontal axis denotes the workloads and the vertical axis denotes EDP value normalized by EDP of a baseline architecture without any parallelism whose parameters are

Table 4.2 – Unrolling factors for CNN-Workloads(M,N,K,S) (loop_m,loop_n,loop_r,loop_c,loop_i,loop_j).

Workloads	Systolic	2Dmapping	Tiling	Selection
LN5_C1(6,1,5,28)	1,1,1,1,5,5	1,1,28,28,1,1	6,1,1,1,1,1	1,1,15,15,5,5
LN5_C3(16,6,5,10)	1,1,1,1,5,5	1,1,10,10,1,1	16,6,1,1,1,1	2,2,7,7,5,5
FR_C1(4,1,5,28)	1,1,1,1,5,5	1,1,28,28,1,1	4,1,1,1,1,1	1,1,15,15,5,5
FR_C3(16,4,4,10)	1,1,1,1,4,4	1,1,10,10,1,1	16,4,1,1,1,1	1,1,10,10,4,4
HG_C1(6,1,5,24)	1,1,1,1,5,5	1,1,24,24,1,1	6,1,1,1,1,1	1,1,16,16,5,5
HG_C3(12,6,4,8)	1,1,1,1,4,4	1,1,8,8,1,1	12,6,1,1,1,1	1,1,7,7,4,4

(1,1,1,1,1,1). The columns represent the normalized EDP of the different architectures. The figure shows that the selected architecture always gets better improvement compared to classical architectures.

The different EDP improvements of these architectures, illustrated in Figure 4.8, can be explained by two main reasons: data reuse and use of computing resources. Systolic and 2D-Mapping architectures have a comparable improvement in terms of energy. Systolic has a higher latency than 2D-Mapping because of the long initialization phase to fill the chain of processing elements. But systolic has a higher data reuse factor than 2D-Mapping, therefore systolic consumes less energy than 2D-Mapping for most workloads. At the SoC level, most of the energy is consumed by the data movement, so if data reuse increases, EDP also increases. In the case of tiling, the EDP improvement is very low because of low computing resource utilization and the poorest energy efficiency due to high latency and poor data reuse. Our selected configuration combines systolic and 2D-Mapping. This corresponds to configurations having maximal synapse parallelism to increases data reuse, and high neuron parallelism to balance computing resource utilization and local memory load/store power consumption.

In summary, the configuration proposed with our flow allows obtaining a better EDP than usual architectures (Systolic, 2D-mapping, and Tiling) for accelerator-based SoCs. This results in an improvement of the EDP by a factor between 2 and 4 compared to a sequential architecture.

4.4.2 Multi-context accelerator

We have found that the optimal hardware accelerators for various CNN applications are rather different from one another. In this part, we investigate if additional optimiza-

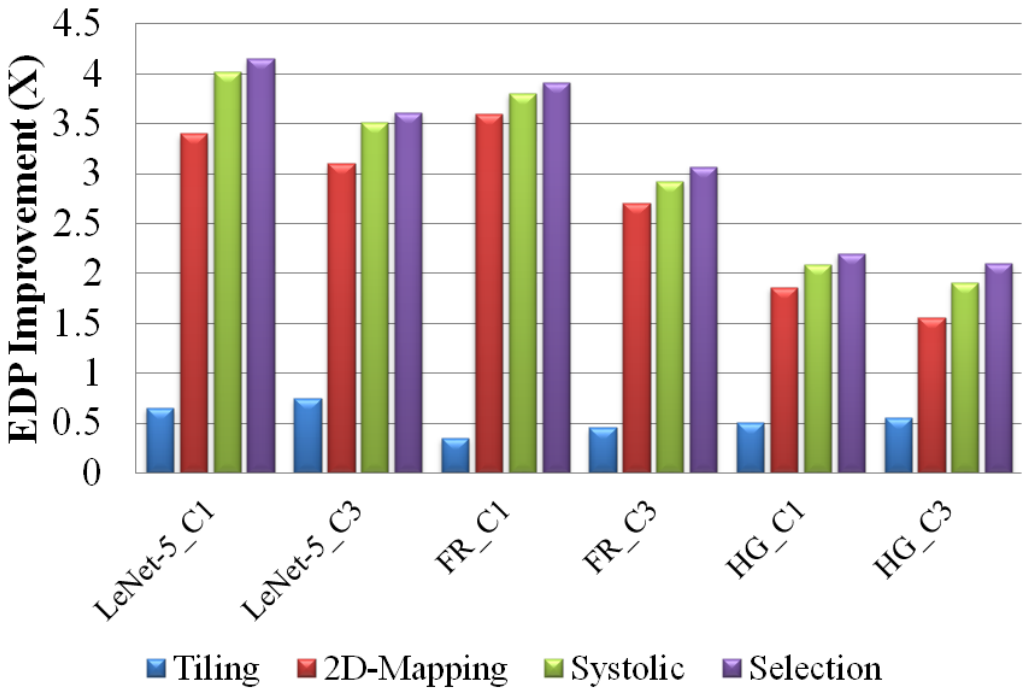


Figure 4.8 – EDP improvement for CNN workloads.

tion possibilities exist when multiple CNN programs run on the same hardware accelerator. First, interleaving the convolutional layers allows us to combine LeNet-5, FT, and HG into multi-context applications. Then, we explore the multi-context CNN-accelerator SoC architectures as well as parallelism and memory optimization using the automatic design flow. EDP is chosen as the optimization metric for a high-performance, low-power accelerator. As a consequence of this, rather than utilizing specific accelerators, we are able to obtain an optimal accelerator that is capable of running a variety of CNN applications.

Our approach to the optimization of the architectural task is to model it as a multidimensional optimization problem. The variables are depicted in Table 4.3. The parallel architectures are constructed using the four loop unrolling factors, where M, N, K, and S represent four parameters at the CONV layer C1/C3 and x::y::z represents a set of values ranging from x to z with a step size of y between each value. For instance, the parameters for $CacheL1Size$ [4::2::128] are 4, 8, 16, 32, 64, and 128. Moreover, since the memory system has a significant impact on the performance of accelerators, we also optimize the architecture that employs a memory hierarchy model, including L1

size, L2 size, and cache configuration.

Table 4.3 – SoC-Accelerators Design Space, where (M,N,K,S) represents four parameters at the CONV layer C1/C3 ,and x::y::z denotes a set of values from x to z by a stepping factor of y.

Parameters	**Values**
Loop Rolling Factor m	[1 $\rightarrow$ max (M_C1, M_C3)]
Loop Rolling Factor n	[1 $\rightarrow$ max (N_C1, N_C3)]
Loop Rolling Factor r_c	[1 $\rightarrow$ max (S_C1, S_C3)]
Loop Rolling Factor i_j	[1 $\rightarrow$ max (K_C1, K_C3)]
Cache L1 Bandwidth (bytes/cycle)	[4::2::128]
Cache L1 Size (KB)	[4::2::128]
Cache L2 Size (KB)	[4::2::256]
Cache Assoc	[4::2::16]

Table 4.4 – Energy Delay Product improvements of the multi-context architecture over the most optimal configuration for each CNN-workload.

Over optimal Lenet5	**Over optimal HG**	**Over optimal FR**
13.4%	10.2%	11.9%

The resulting architectural configurations, along with the EDP that is most suitable for this multi-context CNN accelerator, are displayed in Figure 4.9. This radar chart was created so that a comparison could be made between the various configurations of standalone CNN applications. It is measured against the architectural configurations that achieve the highest effective data processing (EDP) on each CNN workload. According to the data presented in Table 4.4, the multi-context accelerator has an EDP efficiency that is 10.2 percent to 13.4 percent higher than that of the best configuration for each CNN. Since each of the CNN workloads has its own characteristics, the design space is characterized by a wide variety of different configurations. For this reason, consideration of the target applications ought to take place during the early stages of design in order to design effective accelerators for a wide variety of CNN applications. The results of the exploration can be used by the designer to gain an understanding of the steps they should take to mitigate any drawbacks posed by the architecture.

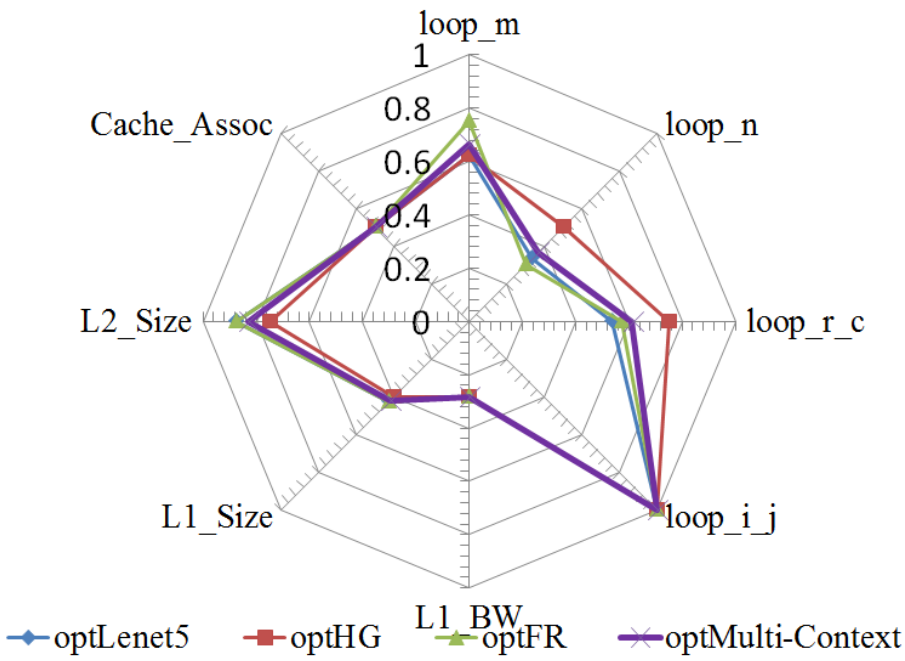


Figure 4.9 – Radar charts of architecture configurations with optimal EDP.

4.4.3 Coherency interface choice study

The SoC configuration used for evaluating heterogeneous-accelerator architectures is a tiled architecture consisting of one CPU and six accelerator tiles, along with L2 cache controller and main memory controller tiles. The processing units all perform different tasks, which means that all the accelerators operate in parallel.

Table 4.5 – Accelerated-workloads in a SoC.

Workloads	**Description**
LeNet5_C1	Convolutional layer (5x5), 32x32 input, 6x28x28 output
LeNet5_C3	Convolutional layer (5x5), 28x28 input, 16x10x10 output
AES-256	AES encryption 256 bits
GEMM_nCubed	Matrix multiplication, 64x64 input
FFT-Transpose	Fast Fourier transform (512-point)
Stencil-3D	Stencil computation, 32x32x16 input
SPMV-Crs	Sparse matrix-vector multiply (2048x512 matrix)

Table 4.5 gives the features of the accelerated workloads used for the experiment. Two LeNet-5 convolutional neural network layers perform an image classification task.

The others correspond to four benchmarks from MachSuite[98]: AES-256, GEMM-nCubed, FFT-Transpose, and Stencil-3D.

This experiment aims to determine the best coherency interface for each of the accelerators separately and for the SoC made up of these six accelerators. The performance of each accelerator is affected not only by its computation time and memory access patterns but also by possible conflicts when accessing shared resources. Consequently, the coherency models adapted to each accelerator are difficult to predict at design time. The input space of hyperparameter is six dimensional due to the six accelerators. Each accelerator interface can be either non-coherent, LLC-coherent or fully-coherent, this results in a total of 729 possible configurations.

Figure 4.10 shows the EDP results for each accelerator and the six-accelerator version. In this figure, the horizontal axis denotes the different accelerators, and the vertical axis denotes EDP normalized with respect to the non-coherent configuration. For each benchmark , the columns represent the normalized EDP of the different interfaces.

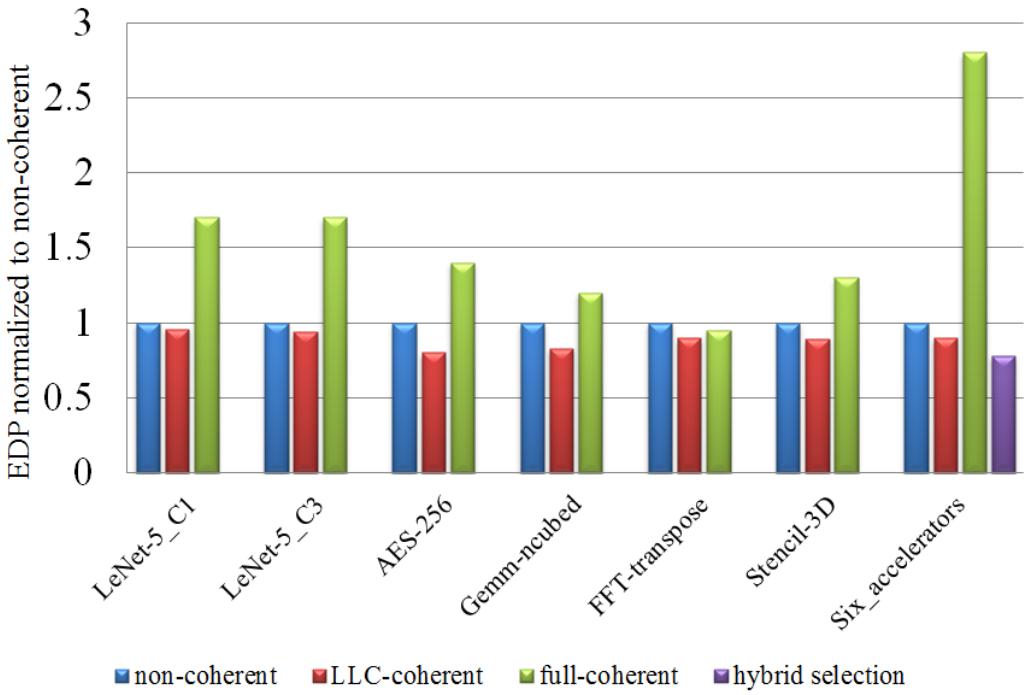


Figure 4.10 – EDP improvement for coherency interface.

In most cases, except for FFT-transpose, the full-coherent interface performs worst

due to its significant hardware and performance overheads. In particular, for CONV-accelerators such as LeNet-5_C1 and LeNet-5_C3. They access a large amount of data (kernels, inputs, and outputs), which cannot fit in the L1 caches and can therefore lead to significant cache misses, penalizing the overall latency. FFT-transpose performs better with fully-coherent than with non-coherent because only eight bytes per 512 bytes of data are read per iteration whereas with the DMA system almost all the data must be available before the computation starts. Furthermore, LLC-coherent shows a better EDP than non-coherent since the memory requests are first sent to the LLC, and when the LLC hits, this results in much shorter access latency.

For the six-accelerator version, the hybrid selection offers better EDP than systems using a single coherency interface. The solution obtained is the following: LeNet-5_C1 and LeNet-5_C3 use non-coherent while the other accelerators use LLC-coherent interface. This hybrid solution results in an improvement in EDP of 22% and 12% respectively, compared to only non-coherent and LLC-coherent.

Although the average (geometric mean) of the EDP improvement over the six accelerators gives the benefit to the only LLC-coherent model, it appears that in the global system view, the hybrid coherent model achieves better EDP improvement.

There are many reasons that explain the EDP improvement brought by the hybrid solution. Having a subset of accelerators with non-coherent interfaces reduces pressure at the LLC level. Indeed, if only four accelerators share the last-level cache, the time spent in data movement compared to an all LLC-coherent solution is reduced. In addition, CONV accelerators benefit from the use of a non-coherent interface with streaming data access patterns applications.

4.4.4 Hyperopt convergence study

We studied the convergence of the hyperparameter optimization algorithm and compared three implementations: random search, conventional TPE, and ATPE. LeNet-5 workload is used as a case study. The three implementations are executed in the same search space and the convergence results are illustrated in Figure 4.11.

The total number of possible configurations is 840. The simulation of all possible configurations confirmed the solution obtained with our optimization method. As illustrated in Figure 4.11, the optimal solution is obtained after a small number of iterations, since 40 are sufficient. The EDP improvement values are distributed into four groups.

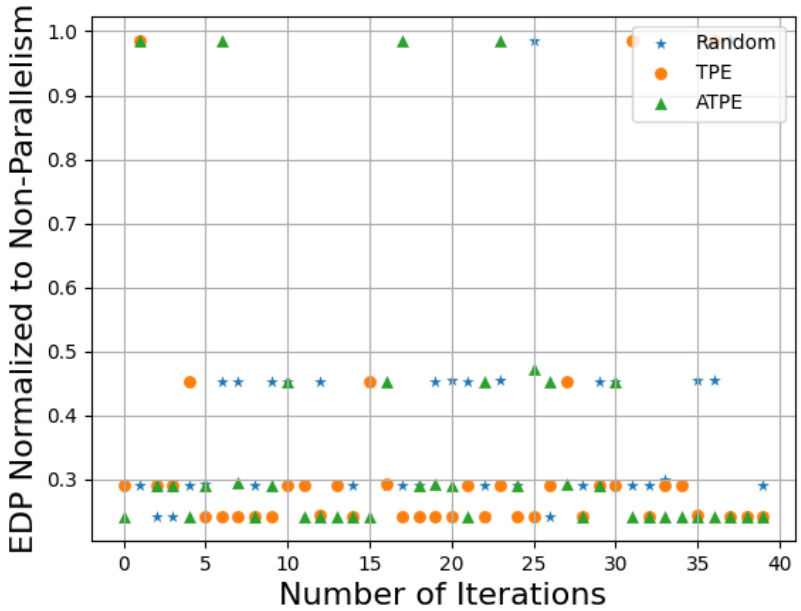


Figure 4.11 – The convergence of parallel exploration in LeNet-5 workload.

This distribution can be explained by the complexity of the exploration space, since we try to mix three types of parallelism, as mentioned in 4.4.1. The ATPE algorithm requires around 30 iterations to converge in the lowest group and achieves the best EDP improvement after 40 iterations. Each iteration requires 30 minutes of CPU time (Intel Xeon E5-2609 at 1.9GHz), considering that gem5-Aladdin represents most of the CPU time.

Figure 4.12 illustrates the convergence study of the coherence interfaces developed for the six accelerators utilizing the automated HyperOpt-gem5-Aladdin framework, from which the lowest EDP normalized to a non-coherent interface was selected. In this instance, 10 random samples were used in the initialization phase, followed by 70 optimization rounds to reach convergence. The ATPE algorithm is found to be effective because it only requires 70 gem5-Aladdin evaluations to get the expected optimal solutions, while there are 729 possible configurations in the whole design space. There are three distinct possibilities for the coherence of each of the six accelerators. On the gem5-Aladdin run, the duration of each iteration of the tool is sixty minutes.

We are able to get a solution more quickly by using TPE and ATPE, as is demonstrated in these examples, which is helpful in that it narrows down the search for good

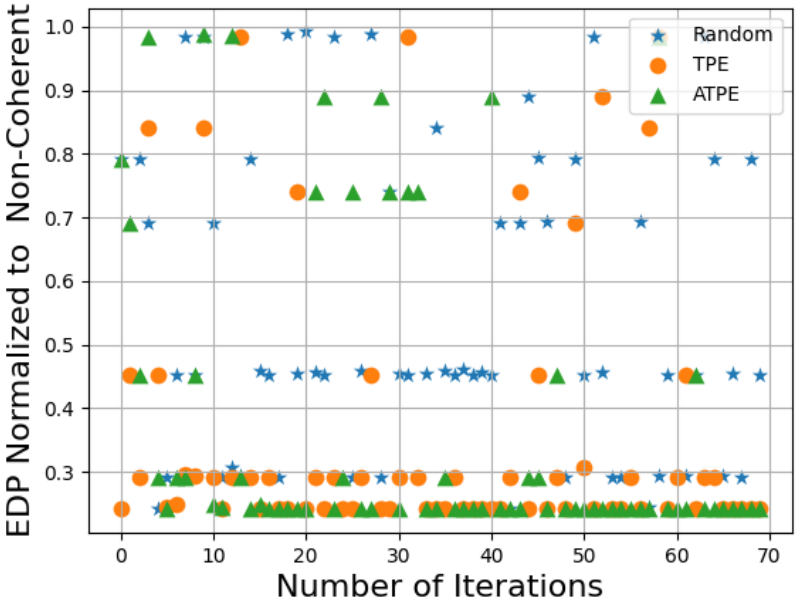


Figure 4.12 – The convergence of the coherency interface experiment.

regions. Compared to traditional TPE, the ATPE demonstrates greater efficiency since it converges faster. Experiments have shown that using this optimized method can significantly decrease the amount of time spent exploring. In the presence of complex design spaces, it proves to be an extremely useful tool.

4.5 Conclusion

In this chapter, we described a flow helping in the design of heterogeneous SoCs. The flow combines the gem5-Aladdin simulator and a hyperparameter optimization method. It automatically identifies the optimal architecture for heterogeneous-accelerator SoCs. To evaluate our approach, we explored the design space of accelerators for convolutional neural networks, including their memory coherency interfaces.

In the case of the CNN experiments, the optimal solution enables a 2x to 4x improvement in the EDP in comparison to an architecture that does not exploit parallelism. In addition to this, its effectiveness surpasses that of the majority of the architectures that are currently in use (systolic, 2D-mapping, and tiling). We optimized one

accelerator for three CNNs and chose the best EDP. The CNN accelerator configuration improves the architecture by 10 to 13.4 percent. We also demonstrated that the use of a hybrid coherency interface enables an improvement in EDP of 22 percent and a 12 percent EDP in comparison to only having a non-coherent or an LLC-coherent interface for a SoC that contains six accelerators. This is in comparison to only having an LLC-coherent interface. From the result, our method can help with large design spaces by reducing the time it takes to explore and by optimizing complex architectural parameters.

CONCLUSION AND FUTURE WORKS

Thesis summary

Processor architecture has moved from homogeneous designs to more complex heterogeneous systems taking advantage of the large number of available transistors by adding customized energy-efficient accelerators as well as general-purpose processor cores. These heterogeneous SoCs are orders of magnitude more efficient and utilize less power than general-purpose processors. Unfortunately, the primary issue with these heterogeneous systems is that they are far more challenging to design and evaluate. This is especially important at a time when customers expect more powerful technological devices in ever-decreasing amounts of time. Therefore, efficient design approaches are necessary to build and explore these complex systems more rapidly.

When considering designing these heterogeneous multi-core systems, the number of possible design combinations leads to a large design space, with often subtle trade-offs. Determining the best design for a given target application requires detailed simulation of many possible solutions. Simulation environments, like gem5, exist and are commonly used to perform these simulations. Unfortunately, these are purely software-based approaches, and they do not allow true exploration of the design space. Moreover, they do not really support heterogeneous multicore architectures (hardware accelerators and processor cores). These limitations motivate the use of specific hardware to accelerate the simulation, and in particular reprogrammable FPGA components.

The first part of the work presented in this dissertation is of an experimental nature. It was about the study of a design approach for heterogeneous architectures based on the design of performance models for the components of the heterogeneous architecture, namely hardware accelerators and processor cores. The contribution focused on the experimentation and evaluation of simulation tools for these heterogeneous architecture models on FPGAs. A methodology for building accelerator performance models and a design flow have been proposed.

The second part of the work carried out is of a methodological nature. It focused

on the study of a flow to determine at the system level a microarchitecture offering the best efficiency in terms of performance/energy consumption ratio. The proposed flow combines two techniques: the use of a software architecture simulator and a hyper-parameter optimization method. This methodology makes it possible to scan different types of parallelism with various loop unwinding strategies while also taking into account different types of interfaces with memories. Experiments on different problems (convolutional neural network, SoC consisting of several accelerators) have made it possible to determine the most optimal solutions in terms of performance/energy consumption ratio.

Future works

There is an element that has not been addressed in this work, and which concerns the integration of the architecture exploration methodology using HyperOpt with the accelerated simulation environment HAsim. Moreover, there is a need to develop a heterogeneous multi-core architecture by merging multi-processor, multi-accelerator, and shared-memory systems with an on-chip network. *This would better distribute processing and acceleration across multiple cores*. It would also be interesting to work on memory structure and memory access patterns for accelerators. Finally, in order to cover the design space more widely, additional parameters (scratchpad partitioning, system bus width, cache size, on-chip network, etc.) would be added. Moreover, it would be interesting to quantify the efficiency of the optimization algorithms and also to integrate new algorithms in order to be able to compare them.

The continued crisis of transistor scaling forces us to think about future architectures. The design community must make an ever greater effort to develop new architectures, design tools, and programming paradigms in response to these challenges. This dissertation describes methods for simulating and exploring heterogeneous architectures at the system level. Thus, further research is needed to make this methodology more fast and cost-effective.

By continuing to integrate others' works into the design infrastructure, I believe that my work can contribute to developing a platform to conduct future research and build heterogeneous systems that scale to larger architectures. This will not only benefit my own research but also future and ongoing research.

BIBLIOGRAPHY

[1] Gene M. Amdahl, « Validity of the Single Processor Approach to Achieving Large Scale Computing Capabilities », *in*: *Proceedings of the April 18-20, 1967, Spring Joint Computer Conference*, AFIPS '67 (Spring), Atlantic City, New Jersey: Association for Computing Machinery, 1967, 483–485, ISBN: 9781450378956, DOI: 10.1145/1465482.1465560, URL: https://doi.org/10.1145/1465482.1465560.

[2] Hari Angepat, Derek Chiou, Eric S. Chung, and James C. Hoe, *FPGA-Accelerated Simulation of Computer Systems*, en, Morgan & Claypool Publishers, July 2014, ISBN: 9781627052146.

[3] Hari Angepat, Dam Sunwoo, and Derek Chiou, « RAMP-White: An FPGA-Based Coherent Shared Memory Parallel Computer Emulator », *in*: Mar. 2007.

[4] *Apple A11 Bionic chip image TMHS09*, URL: https://www.chiprebel.com/apple-a11-bionic/ (visited on 09/16/2020).

[5] Giuseppe Ascia, Vincenzo Catania, Alessandro G. Di Nuovo, Maurizio Palesi, and Davide Patti, « Efficient design space exploration for application specific systems-on-a-chip », *in*: *Journal of Systems Architecture* 53.*10* (2007), Embedded Computer Systems: Architectures, Modeling, and Simulation, pp. 733–750, ISSN: 1383-7621, DOI: https://doi.org/10.1016/j.sysarc.2007.01.004, URL: https://www.sciencedirect.com/science/article/pii/S1383762107000173.

[6] Jonathan Bachrach, Huy Vo, Brian Richards, Yunsup Lee, Andrew Waterman, Rimas Avižienis, John Wawrzynek, and Krste Asanović, « Chisel: Constructing hardware in a Scala embedded language », *in*: *DAC Design Automation Conference 2012*, 2012, pp. 1212–1221, DOI: 10.1145/2228360.2228584.

[7] Fabrice Bellard, « QEMU, a Fast and Portable Dynamic Translator », *in*: *Proceedings of the Annual Conference on USENIX Annual Technical Conference*, ATEC '05, Anaheim, CA: USENIX Association, 2005, pp. 41–41, URL: http://dl.acm.org/citation.cfm?id=1247360.1247401.

[8] J. Bergstra, D. Yamins, and D. D. Cox, « Making a Science of Model Search: Hyperparameter Optimization in Hundreds of Dimensions for Vision Architectures », *in*: *Proceedings of the 30th International Conference on International Conference on Machine Learning - Volume 28*, ICML'13, Atlanta, GA, USA: JMLR.org, 2013, I–115–I–123.

[9] James Bergstra, R. Bardenet, Yoshua Bengio, and Balázs Kégl, « Algorithms for Hyper-Parameter Optimization », *in*: *25th Annual Conference on Neural Information Processing Systems (NIPS 2011)*, Granada, Spain, 2011, URL: https://hal.inria.fr/hal-00642998.

[10] James Bergstra and Yoshua Bengio, « Random search for hyper-parameter optimization. », *in*: *Journal of machine learning research* 13.*2* (2012).

[11] James Bergstra, Brent Komer, Chris Eliasmith, Dan Yamins, and David D. Cox, « Hyperopt: a Python library for model selection and hyperparameter optimization », *in*: *Computational Science and Discovery* 8.*1*, 014008 (Jan. 2015), p. 014008, DOI: 10.1088/1749-4699/8/1/014008.

[12] K. Bhardwaj, M. Havasi, Y. Yao, D. M. Brooks, J. M. H. Lobato, and G. Wei, « Determining Optimal Coherency Interface for Many-Accelerator SoCs Using Bayesian Optimization », *in*: *IEEE Computer Architecture Letters* 18.*2* (2019), pp. 119–123.

[13] David Biancolin, « Automated, FPGA-Based Hardware Emulation of Dynamic Frequency Scaling », PhD thesis, EECS Department, University of California, Berkeley, 2022, URL: http://www2.eecs.berkeley.edu/Pubs/TechRpts/2022/EECS-2022-21.html.

[14] Nathan Binkert et al., « The Gem5 Simulator », *in*: *SIGARCH Comput. Archit. News* 39.*2* (Aug. 2011), pp. 1–7, DOI: 10.1145/2024716.2024718, URL: http://doi.acm.org/10.1145/2024716.2024718 (visited on 11/26/2014).

[15] B. Bohnenstiehl, A. Stillmaker, J. J. Pimentel, T. Andreas, B. Liu, A. T. Tran, E. Adeagbo, and B. M. Baas, « KiloCore: A 32-nm 1000-Processor Computational Array », *in*: *IEEE Journal of Solid-State Circuits* 52.*4* (2017), pp. 891–902, ISSN: 0018-9200, DOI: 10.1109/JSSC.2016.2638459.

[16] João M.P. Cardoso, José Gabriel F. Coutinho, and Pedro C. Diniz, « Chapter 8 - Additional topics », *in*: *Embedded Computing for High Performance*, ed. by João M.P. Cardoso, José Gabriel F. Coutinho, and Pedro C. Diniz, Boston: Morgan Kaufmann, 2017, pp. 255–280, ISBN: 978-0-12-804189-5, DOI: https://doi.org/10.1016/B978-0-12-804189-5.00008-9, URL: https://www.sciencedirect.com/science/article/pii/B9780128041895000089.

[17] Srimat Chakradhar, Murugan Sankaradas, Venkata Jakkula, and Srihari Cadambi, « A Dynamically Configurable Coprocessor for Convolutional Neural Networks », *in*: *Proceedings of the 37th Annual International Symposium on Computer Architecture*, ISCA '10, Saint-Malo, France: Association for Computing Machinery, 2010, 247–257, ISBN: 9781450300537, DOI: 10.1145/1815961.1815993, URL: https://doi.org/10.1145/1815961.1815993.

[18] Tianshi Chen, Zidong Du, Ninghui Sun, Jia Wang, Chengyong Wu, Yunji Chen, and Olivier Temam, « DianNao: A Small-Footprint High-Throughput Accelerator for Ubiquitous Machine-Learning », *in*: *Proceedings of the 19th International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS '14, Salt Lake City, Utah, USA: Association for Computing Machinery, 2014, 269–284, ISBN: 9781450323055, DOI: 10.1145/2541940.2541967, URL: https://doi.org/10.1145/2541940.2541967.

[19] Y. Chen, J. Cong, M. A. Ghodrat, M. Huang, C. Liu, B. Xiao, and Y. Zou, « Accelerator-rich CMPs: From concept to real hardware », *in*: *2013 IEEE 31st International Conference on Computer Design (ICCD)*, 2013, pp. 169–176, DOI: 10.1109/ICCD.2013.6657039.

[20] Y. Chen, J. Cong, and B. Xiao, « ARACompiler: a prototyping flow and evaluation framework for accelerator-rich architectures », *in*: *2015 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, 2015, pp. 157–158.

[21] D. Chiou, H. Angepat, N. Patil, and Dam Sunwoo, « Accurate Functional-First Multicore Simulators », *in*: *Computer Architecture Letters* 8.*2* (Feb. 2009), pp. 64–67, DOI: 10.1109/L-CA.2009.44.

[22] D. Chiou, Dam Sunwoo, H. Angepat, Joonsoo Kim, N.A. Patil, W. Reinhart, and D.E. Johnson, « Parallelizing computer system simulators », *in*: *IEEE International Symposium on Parallel and Distributed Processing, 2008. IPDPS 2008*, Apr. 2008, pp. 1–5, DOI: 10.1109/IPDPS.2008.4536407.

[23] D. Chiou, Dam Sunwoo, Joonsoo Kim, N. Patil, W.H. Reinhart, D.E. Johnson, and Zheng Xu, « The FAST methodology for high-speed SoC/computer simulation », *in*: *IEEE/ACM International Conference on Computer-Aided Design, 2007. ICCAD 2007*, Nov. 2007, pp. 295–302, DOI: 10.1109/ICCAD.2007.4397280.

[24] Derek Chiou, Dam Sunwoo, Joonsoo Kim, Nikhil A. Patil, William Reinhart, Darrel Eric Johnson, Jebediah Keefe, and Hari Angepat, « FPGA-Accelerated Simulation Technologies (FAST): Fast, Full-System, Cycle-Accurate Simulators », *in*: *Proceedings of the 40th Annual IEEE/ACM International Symposium on Microarchitecture*, 2007, pp. 249–

261, DOI: 10.1109/MICRO.2007.36, URL: http://dx.doi.org/10.1109/MICRO.2007.36.

[25] Eric S. Chung, Eriko Nurvitadhi, James C. Hoe, Babak Falsafi, and Ken Mai, « A Complexity-effective Architecture for Accelerating Full-system Multiprocessor Simulations Using FPGAs », *in*: *Proceedings of the 16th International ACM/SIGDA Symposium on Field Programmable Gate Arrays*, 2008, pp. 77–86, DOI: 10.1145/1344671.1344684.

[26] Eric S. Chung, Michael K. Papamichael, Eriko Nurvitadhi, James C. Hoe, Ken Mai, and Babak Falsafi, « ProtoFlex: Towards Scalable, Full-System Multiprocessor Simulations Using FPGAs », *in*: *ACM Trans. Reconfigurable Technol. Syst.* 2.*2* (June 2009), 15:1–15:32, DOI: 10.1145/1534916.1534925, URL: http://doi.acm.org/10.1145/1534916.1534925.

[27] J. Cong, Z. Fang, M. Gill, and G. Reinman, « PARADE: A cycle-accurate full-system simulation Platform for Accelerator-Rich Architectural Design and Exploration », *in*: *2015 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, 2015, pp. 380–387, DOI: 10.1109/ICCAD.2015.7372595.

[28] J. Cong, M. A. Ghodrat, M. Gill, B. Grigorian, and G. Reinman, « Architecture support for accelerator-rich CMPs », *in*: *DAC Design Automation Conference 2012*, 2012, pp. 843–849, DOI: 10.1145/2228360.2228512.

[29] Jason Cong, Mohammad Ali Ghodrat, Michael Gill, Beayna Grigorian, Karthik Gururaj, and Glenn Reinman, « Accelerator-rich architectures: Opportunities and progresses », *in*: *2014 51st ACM/EDAC/IEEE Design Automation Conference (DAC)*, 2014, pp. 1–6, DOI: 10.1145/2593069.2596667.

[30] Jason Cong, Mohammad Ali Ghodrat, Michael Gill, Beayna Grigorian, and Glenn Reinman, « CHARM: A Composable Heterogeneous Accelerator-Rich Microprocessor », *in*: *Proceedings of the 2012 ACM/IEEE International Symposium on Low Power Electronics and Design*, ISLPED '12, Redondo Beach, California, USA: Association for Computing Machinery, 2012, 379–384, ISBN: 9781450312493, DOI: 10.1145/2333660.2333747, URL: https://doi.org/10.1145/2333660.2333747.

[31] Jason Cong and Bingjun Xiao, « Minimizing Computation in Convolutional Neural Networks », *in*: *Artificial Neural Networks and Machine Learning – ICANN 2014*, 2014, pp. 281–290, ISBN: 978-3-319-11179-7.

[32] George E. Dahl, Tara N. Sainath, and Geoffrey E. Hinton, « Improving deep neural networks for LVCSR using rectified linear units and dropout », *in*: *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*, 2013, pp. 8609–8613, DOI: 10.1109/ICASSP.2013.6639346.

[33] S. Davidson et al., « The Celerity Open-Source 511-Core RISC-V Tiered Accelerator Fabric: Fast Architectures and Design Methodologies for Fast Chips », *in*: *IEEE Micro* 38.*2* (2018), pp. 30–41, ISSN: 0272-1732, DOI: 10.1109/MM.2018.022071133.

[34] S. A. Dawwd and B. S. Mahmood, « A reconfigurable interconnected filter for face recognition based on convolution neural network », *in*: *2009 4th International Design and Test Workshop (IDT)*, 2009, pp. 1–6.

[35] R. H. Dennard, F. H. Gaensslen, H. Yu, V. L. Rideout, E. Bassous, and A. R. LeBlanc, « Design of ion-implanted MOSFET's with very small physical dimensions », *in*: *IEEE Journal of Solid-State Circuits* 9.*5* (1974), pp. 256–268.

[36] R. G. Dreslinski, M. Wieckowski, D. Blaauw, D. Sylvester, and T. Mudge, « Near-Threshold Computing: Reclaiming Moore's Law Through Energy Efficient Integrated Circuits », *in*: *Proceedings of the IEEE* 98.*2* (2010), pp. 253–266.

[37] Z. Du, R. Fasthuber, T. Chen, P. Ienne, L. Li, T. Luo, X. Feng, Y. Chen, and O. Temam, « ShiDianNao: Shifting vision processing closer to the sensor », *in*: *2015 ACM/IEEE 42nd Annual International Symposium on Computer Architecture (ISCA)*, 2015, pp. 92–104.

[38] Lieven Eeckhout, « Computer Architecture Performance Evaluation Methods », *in*: *Computer Architecture Performance Evaluation Methods*, 2010.

[39] Katharina Eggensperger, Matthias Feurer, Frank Hutter, James Bergstra, Jasper Snoek, Holger H. Hoos, and Kevin Leyton-brown, « Towards an empirical foundation for assessing Bayesian optimization of hyperparameters », *in*: *In NIPS Workshop on Bayesian Optimization in Theory and Practice*, 2013.

[40] M. E. S. Elrabaa, A. Hroub, M. F. Mudawar, A. Al-Aghbari, M. Al-Asli, and A. Khayyat, « A Very Fast Trace-Driven Simulation Platform for Chip-Multiprocessors Architectural Explorations », *in*: *IEEE Transactions on Parallel and Distributed Systems* 28.*11* (2017), pp. 3033–3045, ISSN: 1045-9219, DOI: 10.1109/TPDS.2017.2713782.

[41] Joel Emer, Carl Beckmann, and Michael Pellauer, « Awb: The asim architect's workbench », *in*: *3rd Annual Workshop on Modeling, Benchmarking, and Simulation (MoBS 2007)*, 2007.

[42] H. Esmaeilzadeh, E. Blem, R. S. Amant, K. Sankaralingam, and D. Burger, « Dark silicon and the end of multicore scaling », *in*: *2011 38th Annual International Symposium on Computer Architecture (ISCA)*, 2011, pp. 365–376.

[43] Paul Feliot, Julien Bect, and Emmanuel Vazquez, « A Bayesian Approach to Constrained Single- and Multi-Objective Optimization », *in*: *J. of Global Optimization* 67.*1–2* (2017), 97–133, ISSN: 0925-5001, DOI: 10.1007/s10898-016-0427-3, URL: https://doi.org/10.1007/s10898-016-0427-3.

[44] Matthias Feurer and Frank Hutter, « Hyperparameter Optimization », *in*: *Automated Machine Learning: Methods, Systems, Challenges*, ed. by Frank Hutter, Lars Kotthoff, and Joaquin Vanschoren, Cham: Springer International Publishing, 2019, pp. 3–33, ISBN: 978-3-030-05318-5, DOI: 10.1007/978-3-030-05318-5_1, URL: https://doi.org/10.1007/978-3-030-05318-5_1.

[45] K. Fleming, Hsin-Jung Yang, M. Adler, and J. Emer, « The LEAP FPGA Operating System », *in*: *2014 24th International Conference on Field Programmable Logic and Applications (FPL)*, Sept. 2014, pp. 1–8, DOI: 10.1109/FPL.2014.6927488.

[46] Kermin Fleming, Chun-Chieh Lin, Nirav Dave, Arvind, Gopal Raghavan, and Jamey Hicks, « H.264 Decoder: A Case Study in Multiple Design Points », *in*: *2008 6th ACM/IEEE International Conference on Formal Methods and Models for Co-Design*, 2008, pp. 165–174, DOI: 10.1109/MEMCOD.2008.4547707.

[47] Andrei Frumusanu, *The Snapdragon 845 - A Quick Recap - The Samsung Galaxy S9 and S9+ Review: Exynos and Snapdragon at 960fps*, Mar. 26, 2018, URL: https://www.anandtech.com/show/12520/the-galaxy-s9-review/2 (visited on 09/16/2020).

[48] A. Fuchs and D. Wentzlaff, « The Accelerator Wall: Limits of Chip Specialization », *in*: *2019 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, 2019, pp. 1–14.

[49] Adi Fuchs, « Overcoming the Limitations of Accelerator-Centric Architectures with Memoization-Driven Specialization », PhD thesis, Princeton University, 2019.

[50] S. Fytraki and D. Pnevmatikatos, « ReSim, a trace-driven, reconfigurable ILP processor simulator », *in*: *Design, Automation Test in Europe Conference Exhibition, 2009. DATE '09.* Apr. 2009, pp. 536–541, DOI: 10.1109/DATE.2009.5090722.

[51] D. Giri, P. Mantovani, and L. P. Carloni, « Accelerators and Coherence: A SoC Perspective », *in*: *IEEE Micro* 38.*6* (2018), pp. 36–45.

[52] V. Govindaraju, C. Ho, T. Nowatzki, J. Chhugani, N. Satish, K. Sankaralingam, and C. Kim, « DySER: Unifying Functionality and Parallelism Specialization for Energy-Efficient Computing », *in*: *IEEE Micro* 32.*5* (2012), pp. 38–51.

[53] Mark Horowitz, « 1.1 Computing's energy problem (and what we can do about it) », *in*: *2014 IEEE International Solid-State Circuits Conference Digest of Technical Papers (ISSCC)*, 2014, pp. 10–14, DOI: 10.1109/ISSCC.2014.6757323.

[54] *https://sochub.fi/wp-content/uploads/2021/03/Making-of-SoCs-presentation-material-1.pdf*, [Online; accessed 2022-07-24].

[55] Q. Huang, C. Yarp, S. Karandikar, N. Pemberton, B. Brock, L. Ma, G. Dai, R. Quitt, K. Asanovic, and J. Wawrzynek, « Centrifuge: Evaluating full-system HLS-generated heterogenous-accelerator SoCs using FPGA-Acceleration », *in*: *2019 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, 2019, pp. 1–8.

[56] Frank Hutter, Holger H. Hoos, and Kevin Leyton-Brown, « Sequential Model-Based Optimization for General Algorithm Configuration », *in*: *Learning and Intelligent Optimization*, ed. by Carlos A. Coello Coello, Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 507–523, ISBN: 978-3-642-25566-3.

[57] Intel, *Acceleration Stack for Intel® Xeon® CPU with FPGAs Core Cache Interface (CCI-P) Reference Manual*, URL: https://www.intel.com/content/dam/www/programmable/us/en/pdfs/literature/manual/archives/mnl-ias-ccip-1-2.pdf.

[58] *IvyTown Xeon + FPGA : The HARP Program.* [Online; accessed 2022-03-11].

[59] Ahmed Kamaleldin and Diana Göhringer, « AGILER: An Adaptive Heterogeneous Tile-Based Many-Core Architecture for RISC-V Processors », *in*: *IEEE Access* 10 (2022), pp. 43895–43913, DOI: 10.1109/ACCESS.2022.3168686.

[60] S. Karandikar et al., « FireSim: FPGA-Accelerated Cycle-Exact Scale-Out System Simulation in the Public Cloud », *in*: *2018 ACM/IEEE 45th Annual International Symposium on Computer Architecture (ISCA)*, 2018, pp. 29–42, DOI: 10.1109/ISCA.2018.00014.

[61] H. Kaul, M. Anders, S. Hsu, A. Agarwal, R. Krishnamurthy, and S. Borkar, « Near-threshold voltage (NTV) design — Opportunities and challenges », *in*: *DAC Design Automation Conference 2012*, 2012, pp. 1149–1154.

[62] A. Krasnov, A. Schultz, J. Wawrzynek, G. Gibeling, and P.-Y. Droz, « RAMP Blue: A Message-Passing Manycore System in FPGAs », *in*: *International Conference on Field Programmable Logic and Applications, 2007. FPL 2007*, Aug. 2007, pp. 54–61, DOI: 10.1109/FPL.2007.4380625.

[63] Monica D. Lam, Edward E. Rothberg, and Michael E. Wolf, « The Cache Performance and Optimizations of Blocked Algorithms », *in*: *Proceedings of the Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS IV, Santa Clara, California, USA: ACM, 1991, pp. 63–74, ISBN: 0-89791-380-9, DOI: 10.1145/106972.106981, URL: http://doi.acm.org/10.1145/106972.106981.

[64] C. Lattner and V. Adve, « LLVM: a compilation framework for lifelong program analysis transformation », *in*: *International Symposium on Code Generation and Optimization, 2004. CGO 2004.* 2004, pp. 75–86, DOI: 10.1109/CGO.2004.1281665.

[65] *LEAP, https://github.com/LEAP-FPGA/leap-documentation/wiki*, 2015, URL: http://riscv.org/ (visited on 08/30/2015).

[66] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, « Gradient-based learning applied to document recognition », *in*: *Proceedings of the IEEE* 86.*11* (1998), pp. 2278–2324.

[67] Sungjin Lee, Kermin Fleming, Jihoon Park, Keonsoo Ha, Adrian Caulfield, Steven Swanson, . Arvind, and Jihong Kim, « BlueSSD: An Open Platform for Cross-layer Experiments for NAND Flash-based SSDs », *in*: *WARP - 5th Annual Workshop on Architectural Research Prototyping*, ed. by Omar Hammami and Sandra Larrabee, Saint Malo, France, June 2010, URL: https://hal.inria.fr/inria-00494143.

[68] Yunsup Lee et al., « An Agile Approach to Building RISC-V Microprocessors », *in*: *IEEE Micro* 36.*2* (2016), 8–20, ISSN: 0272-1732, DOI: 10.1109/MM.2016.11, URL: https://doi.org/10.1109/MM.2016.11.

[69] Rainer Leupers and Olivier Temam, *Processor and System-on-Chip Simulation*, en, Springer Science & Business Media, Sept. 2010, ISBN: 9781441961754.

[70] H. Lin, M. Hsu, and W. Chen, « Human hand gesture recognition using a convolution neural network », *in*: *2014 IEEE International Conference on Automation Science and Engineering (CASE)*, 2014, pp. 1038–1043.

[71] Hung-Yi Liu, Michele Petracca, and Luca P. Carloni, « Compositional system-level design exploration with planning of high-level synthesis », *in*: *2012 Design, Automation Test in Europe Conference Exhibition (DATE)*, 2012, pp. 641–646, DOI: 10.1109/DATE.2012.6176550.

[72] W. Lu, G. Yan, J. Li, S. Gong, Y. Han, and X. Li, « FlexFlow: A Flexible Dataflow Accelerator Architecture for Convolutional Neural Networks », *in*: *2017 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, 2017, pp. 553–564.

[73] M. Lyons, M. Hempstead, G. Wei, and D. Brooks, « The Accelerator Store framework for high-performance, low-power accelerator-based systems », *in*: *IEEE Computer Architecture Letters* 9.*2* (2010), pp. 53–56, ISSN: 1556-6056, DOI: 10.1109/L-CA.2010.16.

[74] Michael J. Lyons, Mark Hempstead, Gu-Yeon Wei, and David Brooks, « The Accelerator Store: A Shared Memory Framework for Accelerator-Based Systems », *in*: *ACM Trans. Archit. Code Optim.* 8.*4* (Jan. 2012), ISSN: 1544-3566, DOI: 10.1145/2086696.2086727, URL: https://doi.org/10.1145/2086696.2086727.

[75] P.S. Magnusson, M. Christensson, J. Eskilson, D. Forsgren, G. Hallberg, J. Hogberg, F. Larsson, A. Moestedt, and B. Werner, « Simics: A full system simulation platform », *in*: *Computer* 35.*2* (Feb. 2002), pp. 50–58, ISSN: 0018-9162, DOI: 10.1109/2.982916.

[76] H. Mair et al., « 23.3 A highly integrated smartphone SoC featuring a 2.5GHz octa-core CPU with advanced high-performance and low-power techniques », *in*: *2015 IEEE International Solid-State Circuits Conference - (ISSCC) Digest of Technical Papers*, 2015, pp. 1–3.

[77] Paolo Mantovani, Davide Giri, Giuseppe Di Guglielmo, Luca Piccolboni, Joseph Zuckerman, Emilio G. Cota, Michele Petracca, Christian Pilato, and Luca P. Carloni, « Agile SoC Development with Open ESP », *in*: *Proceedings of the 39th International Conference on Computer-Aided Design*, ICCAD '20, Virtual Event, USA: Association for Computing Machinery, 2020, ISBN: 9781450380263, DOI: 10.1145/3400302.3415753, URL: https://doi.org/10.1145/3400302.3415753.

[78] Giovanni Mariani, Gianluca Palermo, Vittorio Zaccaria, and Cristina Silvano, « OSCAR: An Optimization Methodology Exploiting Spatial Correlation in Multicore Design Spaces », *in*: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 31.*5* (2012), pp. 740–753, DOI: 10.1109/TCAD.2011.2177457.

[79] Carl J. Mauer, Mark D. Hill, and David A. Wood, « Full-System Timing-First Simulation », *in*: *SIGMETRICS Perform. Eval. Rev.* 30.*1* (2002), 108–116, ISSN: 0163-5999, DOI: 10.1145/511399.511349, URL: https://doi.org/10.1145/511399.511349.

[80] Gábor Melis, Chris Dyer, and Phil Blunsom, « On the State of the Art of Evaluation in Neural Language Models », *in*: *CoRR* abs/1707.05589 (2017), arXiv: 1707.05589, URL: http://arxiv.org/abs/1707.05589.

[81] G. E. Moore, « Cramming more components onto integrated circuits, Reprinted from Electronics, volume 38, number 8, April 19, 1965, pp.114 ff. », *in*: *IEEE Solid-State Circuits Society Newsletter* 11.*3* (2006), pp. 33–35.

[82] Luigi Nardi, Artur Souza, David Koeplinger, and Kunle Olukotun, « HyperMapper: a Practical Design Space Exploration Framework », *in*: *2019 IEEE 27th International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS)*, 2019, pp. 425–426, DOI: 10.1109/MASCOTS.2019.00053.

[83] Man Cheuk Ng, Kermin Elliott Fleming, Mythili Vutukuru, Samuel Gross, Arvind, and Hari Balakrishnan, « Airblue: A system for cross-layer wireless protocol development », *in*: *2010 ACM/IEEE Symposium on Architectures for Networking and Communications Systems (ANCS)*, 2010, pp. 1–11.

[84] Rishiyur Nikhil, « Bluespec System Verilog: efficient, correct RTL from high level specifications », *in*: *Second ACM and IEEE International Conference on Formal Methods and Models for Co-Design, 2004. MEMOCODE '04. Proceedings*, June 2004, pp. 69–70, DOI: 10.1109/MEMCOD.2004.1459818.

[85] J. Park, I. Hong, G. Kim, Y. Kim, K. Lee, S. Park, K. Bong, and H. Yoo, « A 646GOPS/W multi-classifier many-core processor with cortex-like architecture for super-resolution recognition », *in*: *2013 IEEE International Solid-State Circuits Conference Digest of Technical Papers*, 2013, pp. 168–169.

[86] M. Pellauer, M. Adler, D. Chiou, and J. Emer, « Soft connections: Addressing the hardware-design modularity problem », *in*: *46th ACM/IEEE Design Automation Conference, 2009. DAC '09*, July 2009, pp. 276–281.

[87] M. Pellauer, M. Adler, M. Kinsy, A. Parashar, and J. Emer, « HAsim: FPGA-Based High-Detail Multicore Simulation Using Time-Division Multiplexing », *in*: *2011 IEEE 17th International Symposium on High Performance Computer Architecture (HPCA)*, Feb. 2011, pp. 406–417, DOI: 10.1109/HPCA.2011.5749747.

[88] M. Pellauer, M. Vijayaraghavan, M. Adler, Arvind, and J. Emer, « Quick Performance Models Quickly: Closely-Coupled Partitioned Simulation on FPGAs », *in*: *IEEE International Symposium on Performance Analysis of Systems and software, 2008. ISPASS 2008*, Apr. 2008, pp. 1–10, DOI: 10.1109/ISPASS.2008.4510733.

[89] Michael Pellauer, Muralidaran Vijayaraghavan, Michael Adler, Arvind, and Joel Emer, « A-Port Networks: Preserving the Timed Behavior of Synchronous Systems for Modeling on FPGAs », *in*: *ACM Trans. Reconfigurable Technol. Syst.* 2.*3* (Sept. 2009), 16:1–16:26, DOI: 10.1145/1575774.1575775, URL: http://doi.acm.org/10.1145/1575774.1575775.

[90] Michael (Michael Ignatius) Pellauer, « HAsim : cycle-accurate multicore performance models on FPGAs », eng, Thesis, Massachusetts Institute of Technology, 2011, URL: http://dspace.mit.edu/handle/1721.1/64584.

[91] Drew Penney and Lizhong Chen, « A Survey of Machine Learning Applied to Computer Architecture Design », *in*: *ArXiv* abs/1909.12373 (2019).

[92] Daniel Petrisko et al., « BlackParrot: An Agile Open-Source RISC-V Multicore for Accelerator SoCs », *in*: *IEEE Micro* 40.*4* (2020), 93–102, ISSN: 0272-1732, DOI: 10.1109/MM.2020.2996145, URL: https://doi.org/10.1109/MM.2020.2996145.

[93] L. Piccolboni, P. Mantovani, G. Di Guglielmo, and L. P. Carloni, « Broadening the exploration of the accelerator design space in embedded scalable platforms », *in*: *2017 IEEE High Performance Extreme Computing Conference (HPEC)*, 2017, pp. 1–7.

[94] Luca Piccolboni, Paolo Mantovani, Giuseppe Di Guglielmo, and Luca Carloni, « COSMOS: Coordination of High-Level Synthesis and Memory Optimization for Hardware Accelerators », *in*: *ACM Transactions on Embedded Computing Systems* 16 (Sept. 2017), pp. 1–22, DOI: 10.1145/3126566.

[95] Christian Pinto, Shivani Raghav, Andrea Marongiu, Martino Ruggiero, David Atienza, and Luca Benini, « GPGPU-Accelerated Parallel and Fast Simulation of Thousand-Core Platforms », *in*: *Proceedings of the 2011 11th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*, 2011, pp. 53–62, DOI: 10.1109/CCGrid.2011.64.

[96] Fred J. Pollack, « New Microarchitecture Challenges in the Coming Generations of CMOS Process Technologies (Keynote Address)(Abstract Only) », *in*: *Proceedings of the 32nd Annual ACM/IEEE International Symposium on Microarchitecture*, MICRO 32, Haifa, Israel: IEEE Computer Society, 1999, p. 2, ISBN: 076950437X.

[97] J. Pyo et al., « 23.1 20nm high-K metal-gate heterogeneous 64b quad-core CPUs and hexa-core GPU for high-performance and energy-efficient mobile application processor », *in*: *2015 IEEE International Solid-State Circuits Conference - (ISSCC) Digest of Technical Papers*, 2015, pp. 1–3.

[98] B. Reagen, R. Adolf, Y. S. Shao, G. Wei, and D. Brooks, « MachSuite: Benchmarks for accelerator design and customized architectures », *in*: *2014 IEEE International Symposium on Workload Characterization (IISWC)*, 2014, pp. 110–119, DOI: 10.1109/IISWC.2014.6983050.

[99] *Samsung Galaxy S9 Teardown*, URL: https://www.techinsights.com/blog/samsung-galaxy-s9-teardown (visited on 09/16/2020).

[100] Luiz Santos, Sandro Rigo, Rodolfo Azevedo, and Guido Araujo, « Electronic System Level Design », *in*: Jan. 2011, pp. 3–10, ISBN: 978-1-4020-9939-7, DOI: 10.1007/978-1-4020-9940-3_1.

[101] Suleyman Savas, Zain Ul-Abdin, and Tomas Nordström, « Designing Domain-Specific Heterogeneous Architectures from Dataflow Programs », *in*: *Computers* 7 (Apr. 2018), p. 27, DOI: 10.3390/computers7020027.

[102] Benjamin Carrion Schafer and Kazutoshi Wakabayashi, « Design Space Exploration Acceleration Through Operation Clustering », *in*: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 29.*1* (2010), pp. 153–157, DOI: 10.1109/TCAD.2009.2035579.

[103] Herman Schmit and Randy Huang, « Dissecting Xeon + FPGA: Why the Integration of CPUs and FPGAs Makes a Power Difference for the Datacenter: Invited Paper », *in*: *Proceedings of the 2016 International Symposium on Low Power Electronics and Design*, ISLPED '16, San Francisco Airport, CA, USA: Association for Computing Machinery, 2016, 152–153, ISBN: 9781450341851, DOI: 10.1145/2934583.2953983, URL: https://doi.org/10.1145/2934583.2953983.

[104] B. Shahriari, K. Swersky, Z. Wang, R. P. Adams, and N. de Freitas, « Taking the Human Out of the Loop: A Review of Bayesian Optimization », *in*: *Proceedings of the IEEE* 104.*1* (2016), pp. 148–175.

[105] Y. S. Shao, B. Reagen, G. Wei, and D. Brooks, « The Aladdin Approach to Accelerator Design and Modeling », *in*: *IEEE Micro* 35.*3* (2015), pp. 58–70, ISSN: 0272-1732, DOI: 10.1109/MM.2015.50.

[106] Y. S. Shao, S. L. Xi, V. Srinivasan, G. Wei, and D. Brooks, « Co-designing accelerators and SoC interfaces using gem5-Aladdin », *in*: *2016 49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2016, pp. 1–12, DOI: 10.1109/MICRO.2016.7783751.

[107] Yakun Sophia Shao and David M. Brooks, *Research Infrastructures for Hardware Accelerators*, Synthesis Lectures on Computer Architecture, Morgan & Claypool Publishers, 2015, DOI: 10.2200/S00677ED1V01Y201511CAC034, URL: https://doi.org/10.2200/S00677ED1V01Y201511CAC034.

[108] Amith Singhee and Pamela Castalino, « Pareto sampling: Choosing the right weights by derivative pursuit », *in*: *Design Automation Conference*, 2010, pp. 913–916, DOI: 10.1145/1837274.1837503.

[109] Jasper Snoek, Hugo Larochelle, and Ryan P. Adams, « Practical Bayesian Optimization of Machine Learning Algorithms », *in*: *Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 2*, NIPS'12, Lake Tahoe, Nevada: Curran Associates Inc., 2012, 2951–2959.

[110] Jasper Snoek, Oren Rippel, Kevin Swersky, Ryan Kiros, Nadathur Satish, Narayanan Sundaram, Md. Mostofa Ali Patwary, Prabhat Prabhat, and Ryan P. Adams, « Scalable Bayesian Optimization Using Deep Neural Networks », *in*: *Proceedings of the 32nd International Conference on International Conference on Machine Learning - Volume 37*, ICML'15, Lille, France: JMLR.org, 2015, 2171–2180.

[111] Synopsys, *HAPS ® Family of FPGA-Based Prototyping Solutions*, http://www.synopsys.com/Systems/FPGABasedPrototyping/Pages/HAPS.aspx, 2014.

[112] Zhangxi Tan, Andrew Waterman, Rimas Avizienis, Yunsup Lee, Henry Cook, David Patterson, and Krste Asanović, « RAMP Gold: An FPGA-based Architecture Simulator for Multiprocessors », *in*: *Proceedings of the 47th Design Automation Conference*, DAC '10, New York, NY, USA: ACM, 2010, pp. 463–468, ISBN: 978-1-4503-0002-5, DOI: 10.1145/1837274.1837390, URL: http://doi.acm.org/10.1145/1837274.1837390 (visited on 11/26/2014).

[113] Zhangxi Tan, Andrew Waterman, Henry Cook, Sarah Bird, Krste Asanović, and David Patterson, « A Case for FAME: FPGA Architecture Model Execution », *in*: *Proceedings of the 37th Annual International Symposium on Computer Architecture*, 2010, pp. 290–301, DOI: 10.1145/1815961.1815999, URL: http://doi.acm.org/10.1145/1815961.1815999.

[114] M. B. Taylor, « Is dark silicon useful? Harnessing the four horsemen of the coming dark silicon apocalypse », *in*: *DAC Design Automation Conference 2012*, 2012, pp. 1131–1136.

[115] *The End of the Road for General Purpose Processors & the Future of Computing | MIT CSAIL*, URL: https://www.csail.mit.edu/news/end-road-general-purpose-processors-future-computing (visited on 10/07/2020).

[116] Srinivasa R Vemuru and Norman Scheinberg, « Short-circuit power dissipation estimation for CMOS logic gates », *in*: *IEEE Transactions on Circuits and Systems I: Fundamental Theory and Applications* 41.*11* (1994), pp. 762–765.

[117] Ganesh Venkatesh, Jack Sampson, Nathan Goulding, Saturnino Garcia, Vladyslav Bryksin, Jose Lugo-Martinez, Steven Swanson, and Michael Bedford Taylor, « Conservation Cores: Reducing the Energy of Mature Computations », *in*: *SIGPLAN Not.* 45.*3* (Mar. 2010), 205–218, ISSN: 0362-1340, DOI: 10.1145/1735971.1736044, URL: https://doi.org/10.1145/1735971.1736044.

[118] Danyao Wang, Natalie Enright Jerger, and J. Gregory Steffan, « DART: A programmable architecture for NoC simulation on FPGAs », *in*: *Proceedings of the Fifth ACM/IEEE International Symposium*, 2011, pp. 145–152.

[119] Andrew Waterman, Yunsup Lee, David A. Patterson, and Krste Asanović, *The RISC-V Instruction Set Manual, Volume I: User-Level ISA, Version 2.0*, tech. rep. UCB/EECS-2014-54, EECS Department, University of California, Berkeley, May 2014, URL: http://www.eecs.berkeley.edu/Pubs/TechRpts/2014/EECS-2014-54.html.

[120] J. Wawrzynek, D. Patterson, M. Oskin, Shih-Lien Lu, C. Kozyrakis, J.C. Hoe, D. Chiou, and K. Asanovic, « RAMP: Research Accelerator for Multiple Processors », *in*: *IEEE Micro* 27.*2* (Mar. 2007), pp. 46–57, DOI: 10.1109/MM.2007.39.

[121] Nan Wu and Yuan Xie, « A Survey of Machine Learning for Computer Architecture and Systems », *in*: *ACM Comput. Surv.* 55.*3* (2022), ISSN: 0360-0300, DOI: 10.1145/3494523, URL: https://doi.org/10.1145/3494523.

[122] Ning Zhang and Bob Brodersen, « The cost of flexibility in systems on a chip design for signal processing applications », *in*: (Jan. 2002).

[123] Zhiru Zhang, Yiping Fan, Wei Jiang, Guoling Han, Changqi Yang, and Jason Cong, « AutoPilot: A platform-based ESL synthesis system », *in*: Jan. 2008, pp. 99–112, ISBN: 9781402085871, DOI: 10.1007/978-1-4020-8588-8.

ACRONYMS

AFU accelerated function unit. 59

AI artificial intelligence. 17

API application programming interface. 59

ASIC application specific integrated circuit. 19

ATPE adaptive tree of parzen estimators. 101

BSV bluespec system verilog. 10, 39

CAD computer-aided design. 26, 27

CCI-P core cache interface. 40

CET compact execution trace. 28

CNN convolutional neural network. 6, 10

CPU central processing unit. 3, 16

CSRs control and status registers. 63

DDDG dynamic data dependence graph. 76

DMA direct memory access. 4, 11

DSP digital signal processing. 2

EDP energy-delay-product. 10, 94

FAME FPGA accelerated model execution. 25, 38

FIFO First In First Out. 61

FIU FPGA interface unit. 59

FPGA field programmable gate array. 6, 19

GP gaussian processes. 98

GPU graphics processing unit. 3, 17

HARP hardware accelerator research program. 39

HAs heterogeneous architectures. 1

HLS high-level synthesis. 27

IL ACE IL academic compute environment. 39, 40

IR intermediate representation. 78

ISA instruction set architecture. 29, 39, 42, 49, 50, 85

JIT just-in-time. 78

LEAP latency-insensitive environment for application programming. 38, 58

LLVM low-level virtual machine. 78

MMIO memory-mapped I/O. 60

NLP natural language processing. 17

NoC network on chip. 17, 25

OPAE open programmable acceleration engine. 40

PAC programmable acceleration card. 59

PCIe peripheral component interconnect express. 56

QoR quality of results. 32

QPI QuickPath interconnect. 59

RAMP research accelerator for multiple processors. 23, 40

RTL register transfer level. 27

SFF speculative functional-first. 23

SMBO sequential model-based optimization. 97

SoC systems-on-chip. 17

SSA static single assignment. 78

TPE tree parzen estimator. 99