

**Miguel Chavarrías
Alfonso Rodríguez (Eds.)**

LNCS 13879

Design and Architecture for Signal and Image Processing

**16th International Workshop, DASIP 2023
Toulouse, France, January 16–18, 2023
Proceedings**

 **Springer**

Lecture Notes in Computer Science

13879

Founding Editors

Gerhard Goos
Juris Hartmanis

Editorial Board Members

Elisa Bertino, *Purdue University, West Lafayette, IN, USA*

Wen Gao, *Peking University, Beijing, China*

Bernhard Steffen , *TU Dortmund University, Dortmund, Germany*

Moti Yung , *Columbia University, New York, NY, USA*

The series Lecture Notes in Computer Science (LNCS), including its subseries Lecture Notes in Artificial Intelligence (LNAI) and Lecture Notes in Bioinformatics (LNBI), has established itself as a medium for the publication of new developments in computer science and information technology research, teaching, and education.

LNCS enjoys close cooperation with the computer science R & D community, the series counts many renowned academics among its volume editors and paper authors, and collaborates with prestigious societies. Its mission is to serve this international community by providing an invaluable service, mainly focused on the publication of conference and workshop proceedings and postproceedings. LNCS commenced publication in 1973.

Miguel Chavarrías · Alfonso Rodríguez
Editors

Design and Architecture for Signal and Image Processing

16th International Workshop, DASIP 2023
Toulouse, France, January 16–18, 2023
Proceedings

Editors

Miguel Chavarrías 
Polytechnic University of Madrid
Madrid, Spain

Alfonso Rodríguez 
Polytechnic University of Madrid
Madrid, Spain

ISSN 0302-9743

ISSN 1611-3349 (electronic)

Lecture Notes in Computer Science

ISBN 978-3-031-29969-8

ISBN 978-3-031-29970-4 (eBook)

<https://doi.org/10.1007/978-3-031-29970-4>

© The Editor(s) (if applicable) and The Author(s), under exclusive license
to Springer Nature Switzerland AG 2023

This work is subject to copyright. All rights are reserved by the Publisher, whether the whole or part of the material is concerned, specifically the rights of translation, reprinting, reuse of illustrations, recitation, broadcasting, reproduction on microfilms or in any other physical way, and transmission or information storage and retrieval, electronic adaptation, computer software, or by similar or dissimilar methodology now known or hereafter developed.

The use of general descriptive names, registered names, trademarks, service marks, etc. in this publication does not imply, even in the absence of a specific statement, that such names are exempt from the relevant protective laws and regulations and therefore free for general use.

The publisher, the authors, and the editors are safe to assume that the advice and information in this book are believed to be true and accurate at the date of publication. Neither the publisher nor the authors or the editors give a warranty, expressed or implied, with respect to the material contained herein or for any errors or omissions that may have been made. The publisher remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.

This Springer imprint is published by the registered company Springer Nature Switzerland AG
The registered company address is: Gewerbestrasse 11, 6330 Cham, Switzerland

Preface

This volume contains papers presented at the 2023 Workshop on Design and Architectures for Signal and Image Processing (DASIP 2023) – 16th edition, which was held jointly with the 18th HiPEAC Conference, January 16–18, 2023 in Toulouse, France. DASIP provides an inspiring international forum for the latest innovations and developments in the field of leading signal, image and video processing and machine learning in custom embedded, edge and cloud computing architectures and systems.

For this edition of the workshop, we received 17 paper submissions from 6 countries around the world, and 9 high-quality papers were accepted as oral presentations. Each contributed paper underwent a rigorous double-blind peer-review process during which it was reviewed by at least three reviewers who were drawn from a large pool of the Technical Program Committee members.

The success of DASIP depends on the contributions of many individuals and organizations. With that in mind, we thank all authors who submitted their work to the conference. We also wish to offer our sincere thanks to the members of the Technical Program Committee for their very detailed reviews, and to the members of the Steering Committee.

We would also like to address special thanks to Tomasz Kryjak, from AGH University of Science and Technology (Poland), for presenting a deeply inspiring keynote during the event.

February 2023

Miguel Chavarrías
Alfonso Rodríguez

Organization

General Chairs

Miguel Chavarrías
Alfonso Rodríguez

Universidad Politécnica de Madrid, Spain
Universidad Politécnica de Madrid, Spain

Steering Committee

Bertrand Granado
Diana Goehringer
Eduardo de la Torre
Guy Gogniat
Jean-François Nezan
Jean-Pierre David
João M. P. Cardoso
Marek Gorgon

Sorbonne Université, France
TU Dresden, Germany
Universidad Politécnica de Madrid, Spain
Université de Bretagne Sud, France
INSA Rennes, France
Polytechnique Montréal, Canada
University of Porto, Portugal
AGH University of Science and Technology,
Poland
Brandenburg University of Technology, Germany
AGH University of Science and Technology,
Poland
Università degli Studi di Cagliari, Italy
Polytechnique Montréal, Canada
University of Nantes - IETR, France

Michael Huebner
Tomasz Kryjak

Paolo Meloni
Pierre Langlois
Sébastien Pillement

Program Committee

Jani Boutellier
Gabriel Caffarena
Gustavo Marrero

University of Vaasa, Finland
University CEU San Pablo, Spain
Universidad de Las Palmas de Gran Canaria,
Spain

João M. P. Cardoso
Miguel Chavarrías
Daniel Chillet
Martin Danek
Jean-Pierre David
Karol Desnos

University of Porto, Portugal
Universidad Politécnica de Madrid, Spain
University of Rennes - IRISA/ENSSAT, France
daiteq s.r.o., Czech Republic
Polytechnique Montréal, Canada
INSA Rennes, France

Milos Drutarovsky	Technical University of Kosice, Slovakia
Himar Fabelo	Universidad de Las Palmas de Gran Canaria, Spain
Diana Goehringer	TU Dresden, Germany
Guy Gogniat	Université de Bretagne Sud, France
Marek Gorgon	AGH University of Science and Technology, Poland
Bertrand Granado	Sorbonne Université, France
Oscar Gustafsson	Linköping University, Sweden
Frank Hannig	Friedrich-Alexander University, Germany
Dominique Houzet	GIPSA-Lab, France
Michael Huebner	Brandenburg University of Technology, Germany
Eduardo Juárez	Universidad Politécnica de Madrid, Spain
Mateusz Komorkiewicz	Aptiv, Poland
Tomasz Kryjak	AGH University of Science and Technology, Poland
Pierre Langlois	Polytechnique Montréal, Canada
Raquel Lazcano	Università degli Studi di Sassari, Italy
Yannick Le Moullec	Tallinn University of Technology, Estonia
Daniel Madroñal	Università degli Studi di Sassari, Italy
Paolo Meloni	Università degli Studi di Cagliari, Italy
Jari Nurmi	Tampere University, Finland
Arnaldo Oliveira	Universidade de Aveiro, Portugal
Andrés Otero	Universidad Politécnica de Madrid, Spain
Maxime Pelcat	INSA Rennes, France
Fernando Pescador	Universidad Politécnica de Madrid, Spain
Christian Pilato	Politecnico di Milano, Italy
Sébastien Pillement	University of Nantes - IETR, France
Andrea Pinna	LIP6 UPMC, France
Alfonso Rodríguez	Universidad Politécnica de Madrid, Spain
Nuno Roma	Universidade de Lisboa, Portugal
Olivier Romain	ETIS, France
Gonzalo Rosa	Universidad Politécnica de Madrid, Spain
Paweł, Russek	AGH University of Science and Technology, Poland
Rubén Salvador	CentraleSupélec - IETR, France
Carlo Sau	Università degli Studi di Cagliari, Italy
Yves Sorel	Inria, France
Dimitrios Soudris	National Technical University of Athens, Greece
Pablo Sánchez	University of Cantabria, Spain
Manuel Villa	Universidad Politécnica de Madrid, Spain
Serge Weber	University of Lorraine, France

Contents

Methods and Applications

SCAPE: HW-Aware Clustering of Dataflow Actors for Tunable Scheduling Complexity	3
<i>Ophélie Renaud, Dylan Gageot, Karol Desnos, and Jean-François Nezan</i>	
Deep Recurrent Neural Network Performing Spectral Recurrence on Hyperspectral Images for Brain Tissue Classification	15
<i>Pedro L. Cebrián, Alberto Martín-Pérez, Manuel Villa, Jaime Sancho, Gonzalo Rosa, Guillermo Vazquez, Pallab Sutradhar, Alejandro Martinez de Ternerero, Miguel Chavarriás, Alfonso Lagares, Eduardo Juarez, and César Sanz</i>	
Brain Blood Vessel Segmentation in Hyperspectral Images Through Linear Operators	28
<i>Guillermo Vazquez, Manuel Villa, Alberto Martín-Pérez, Jaime Sancho, Gonzalo Rosa, Pedro L. Cebrián, Pallab Sutradhar, Alejandro Martinez de Ternerero, Miguel Chavarriás, Alfonso Lagares, Eduardo Juarez, and César Sanz</i>	
Neural Network Predictor for Fast Channel Change on DVB Set-Top-Boxes ...	40
<i>Tomás Malcata, Nuno Sebastião, Tiago Dias, and Nuno Roma</i>	

Hardware Architectures and Implementations

AINoC: New Interconnect for Future Deep Neural Network Accelerators	55
<i>Hana Krichene, Rohit Prasad, and Ayoub Mouhagir</i>	
Real-Time FPGA Implementation of the Semi-global Matching Stereo Vision Algorithm for a 4K/UHD Video Stream	70
<i>Mariusz Grabowski and Tomasz Kryjak</i>	
TaPaFuzz - An FPGA-Accelerated Framework for RISC-V IoT Graybox Fuzzing	82
<i>Florian Meisel, David Volz, Christoph Spang, Dat Tran, and Andreas Koch</i>	

**Adaptive Inference for FPGA-Based 5G Automatic Modulation
Classification** 95
*Daniel de Oliveira Rubiano, Guilherme Korol,
and Antonio Carlos Schneider Beck*

**High-Level Online Power Monitoring of FPGA IP Based on Machine
Learning** 107
*Majdi Richa, Jean-Christophe Prévotet, Mickaël Dardaillon,
Mohamad Mroué, and Abed Ellatif Samhat*

Author Index 121

Methods and Applications



SCAPE: HW-Aware Clustering of Dataflow Actors for Tunable Scheduling Complexity

Ophélie Renaud¹(✉), Dylan Gageot², Karol Desnos¹,
and Jean-François Nezan¹

¹ Univ Rennes, INSA Rennes, CNRS, IETR - UMR 6164, Rennes, France
{ophelie.renaud, karol.desnol, jean-francois.nezan}@insa-rennes.fr

² Yubik, Rennes, France
dgageot@yubik.io

Abstract. This paper introduces a fast method to generate high performance parallelized code from a dataflow specification of an application. Dataflow Models of Computation (MoCs) are efficient programming paradigms for expressing the parallelism of an application. Traditionally, mapping and scheduling methods for dataflow MoCs rely on complex graph's transformations to explicit their parallelism which can result in complex graph for embarrassingly parallel applications. For such applications, state-of-the-art mapping and scheduling techniques are prohibitively complex, while the exposed parallelism often exceeds the parallel processing capabilities of the target architecture. We propose SCAPE, an automated method to control the complexity of the pre-scheduling graph transformation by using information from the architecture and application models. By decreasing the complexity of the graph, the mapping scheduling task is accelerated at the potential expense of the produced schedule. Our method offers a limited and controlled decrease of the schedule quality while enabling mapping and scheduling execution time between 1 and 2 orders of magnitude faster than state-of-the-art techniques.

Keywords: Dataflow model · Hierarchy · Granularity · Clustering

1 Introduction

Digital signal processing technology emerged in the 1960s and has grown rapidly, becoming more complex over the years, particularly with the arrival of machine learning applications a decade ago. To meet the ever-increasing need for computing power and speed of execution of these applications, developers first sought to increase the frequency of individual Processing Elements (PEs) and then turned to heterogeneous multicore embedded systems.

This work was supported by DARK-ERA (ANR-20-CE46-0001-01).

© The Author(s), under exclusive license to Springer Nature Switzerland AG 2023
M. Chavarrías and A. Rodríguez (Eds.): DASIP 2023, LNCS 13879, pp. 3–14, 2023.
https://doi.org/10.1007/978-3-031-29970-4_1

Exploiting in an optimized way the maximum parallelism of such multicore target architectures is very challenging. The development of parallel code is tedious and is not adapted to manage hardware and software upgrades during the exploitation phase of the project. Tools such as Simulink [6] and Xilinx AI Engine Technology [1] are then investigated to automate the rapid deployment of new algorithms on the computing system. Both tools are based on the dataflow approach consisting in the modeling of the algorithms by a graph in which nodes, called actors, represent the calculations and directed arcs, called First In First Out queue (FIFO) buffers, represent the data, called tokens, exchanges between nodes.

The automated generation of parallel code from such dataflow models requires solving several NP-Complete problems, especially for resource allocation. Calculations are distributed on the PEs of the target architecture and will read and write, during the execution of an application, on FIFO buffers assigned to a range of memory addresses. The resource allocation choices can be made at compile time or at runtime. The allocation at runtime leads to performance overhead and unpredictable application behavior. For these reasons, this paper investigates methods that allocate resources at compile time, during software synthesis. The software synthesis process is responsible for translating a dataflow model into a prototype.

Classic resource allocation methods involve three phases: The *mapping* consists in distributing actors on the PEs of the target. The *Scheduling* consists in ordering the execution of actors on the PEs. The *Timing* associates to each actor a start time and an end time, useful to calculate the long time average throughput, also called latency, of the application. The time required for the mapping and scheduling process grows exponentially with the number of PEs and the number of nodes and edges of the dataflow graph [7].

The Scaling up of Clusters of Actors on Processing Element (SCAPE) method is introduced in this paper which is a hierarchy-based clustering method that transforms an application to match its degree of parallelism to the parallel computation capabilities of the targeted architecture. The method offers as many clustering configurations as there are hierarchy levels in the Synchronous Dataflow (SDF) input graph which gives the user the possibility to choose the required granularity for a reduced software synthesis time.

Section 2 presents dataflow MoCs, the traditional mapping and scheduling method and the state-of-the-art clustering heuristics. Section 3 describes the proposed SCAPE method and the backbone of the resulting code. Section 4 outlines experimental results on several granularity clustering configurations showing a tradeoff between design space exploration time and produced schedule latency. Finally, Section 5 concludes this paper.

2 Context and Related Work

2.1 SDF Based Dataflow MoCs

The most studied dataflow MoC is the SDF [8] illustrated in the Fig. 1, in which the integer numbers on the input and output ports of actors are the rate of tokens respectively consumed and produced by actors at each execution of the actors.

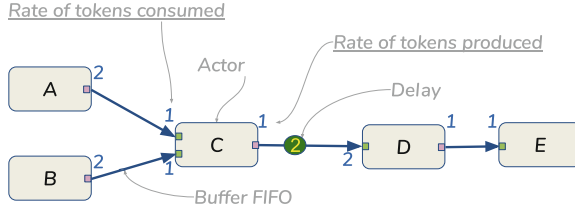


Fig. 1. SDF MoC semantics

An SDF graph is usually transformed into an equivalent Single rate Directed Acyclic Graphs (SrDAG) [12] to map and compute a periodic deadlock-free schedule iterated infinitely. A repetition of a periodic schedule is called a graph iteration. The single rate transformation consists in duplicating the actors by the number of firings specified by the schedule, and in adding special actors to distribute or gather tokens, so that the production and the consumption rates on each FIFO are equals. Then, DAG transformation consists in breaking cyclic data-paths. To be consistent, all cyclic data paths must contain at least one initial data token also called delays. Therefore, breaking a cycles means replacing FIFOs with delays by a pair of special actors which backup delayed tokens and read the backed up tokens.

An extension of the SDF model is the Parameterized and Interfaced SDF (PiSDF) model [4]. In this paper, the feature of interest of the PiSDF model is its support for hierarchy. The hierarchy feature allows the internal behavior of actors to be specified by a subgraph instead of C code. The PiSDF MoC defines interfaces of a subgraph as input and output data ports of the parent hierarchical actor. Interfaces allow the transmission of tokens between hierarchical levels. The hierarchy is used to represent different levels of granularity of the computations that compose an application, the lower levels of hierarchy being the finer granularity.

These SDF based models have two main advantages justifying their interest. The first one is to express the three types of parallelism [13], two of which are used in this paper: task and data parallelism (pipeline being the third).

- Task parallelism is expressed by two actors belonging to parallel data-paths like actors A and B in the Fig. 1. As there is no data-path between these actors, they can be fired at the same time.
- Data parallelism is expressed when several firings of a single actor are independent from each other. If enough data tokens are present in the input FIFOs, then several firings can be executed concurrently. An example is the actor C in the Fig. 1 which can be executed 2 times when A and B are executed 1 time.

The second advantage is that the model is independent of the target architecture. An application is represented once and executed on all types of architecture (single cores, multicores with shared or distributed memories, FPGA, etc.).

2.2 Classic Flattening Method

This paper focuses on tools that have chosen to allocate resources at compile time, also called static allocation. Software synthesis can be modeled by a workflow. The typical static scheduling workflow is composed of four main tasks: flattening, SrDAG transformation, mapping and scheduling, then code generation.

The *flattening* task of the workflow consists in putting all the actors of a graph at the same level which means all hierarchical actors are replaced by their subgraph. To keep the functionality of the application in Fig. 2, token rates consumed and produced in the initial subgraph are scaled up on upper levels. Tools usually flatten the whole graph to execute the rest of the process which brings the finer level of granularity to the top-level graph.

The *SrDAG transformation* task is used to reveal parallelism on flattened graph Fig. 2. It highlights the minimal number of firings of each actor to return the graph back to its original state given by the calculation of the Repetition Vector (RV) \mathbf{q} . Here, actor B is fired 4 times per graph iteration, so $\mathbf{q}(B) = 4$. It also emphasizes the interdependencies between the actors, which is useful to calculate the execution order of the actors, allowing them to be iterated infinitely without generating a deadlock. In the figure, the named nodes are the actor instances and the unnamed nodes are the special actors responsible for distributing or gathering the data tokens.

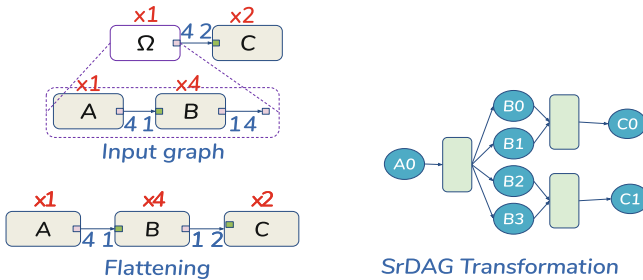


Fig. 2. Classic flattening process: 3 SDF actors turn into 10 SrDAG actors

The excessive complexity of the SrDAG increases the mapping opportunities which results in a better distribution of the computations on the different PEs and reduces the latency of the application on the target. Since the mapping opportunities are limited to the number of PEs in the architecture, exposing more parallelism than the number of PEs is unnecessary and time consuming. [10]. This is why reducing the exposed parallelism to the number of PE will most likely be sufficient to fully exploit the architecture parallelism, while being much simpler to map and schedule.

Example 1. Considering a machine learning application: *Squeezenet* neural network whose SDF model is composed of 70 actors, its SrDAG transformation

results in 5452 actors. Mapping this application on an architecture composed of 8 PEs with a greedy algorithms requires considering and evaluating 8 mapping choices. Here the degree of parallelism is up to 1000 which is an needlessly fine granularity.

2.3 Cluster of SDF Actors

A way to reduce the complexity of mapping and scheduling algorithms is to reduce the number of actors to map in the srDAG, without altering behavior of the application. This reduction can be achieved using clustering techniques, wick transform the input graph by grouping actors with a particular behavior. Since grouping two or more actors into a single equivalent hierarchical actor may change the behavior of the application, or even create deadlocks, clustering rules have been introduced in [10]. These rules are illustrated in Fig. 3 where SDF graphs are represented with rectangular actors and the corresponding precedence graphs with round actors. A cluster must respect the execution order of the actors defined by the precedence rules (a), the initial tokens must be considered (b) and there must be no simple path from a node of the precedence graph to another one that contains more that one arc(c). A simple path is the one which does not visit any node along the path more than once.

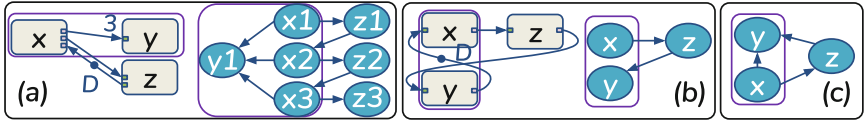


Fig. 3. (a) illustrate the violation of the first precedence shift condition, (b) illustrate the violation of the hidden delay condition, and (c) illustrate the violation of the cycle introduction condition

A method to cluster SDF actors is presented in [2] which introduced the Pairwise Grouping of Adjacent Nodes for Acyclic graph (APGAN) algorithm. Considering an acyclic SDF graph $G = \langle A, F \rangle$ where A is a set of stateless actors and F is a set of FIFO, the algorithm can be summarized as follows: A cluster hierarchy is constructed by clustering exactly two adjacent vertices at each step. At each clusterization step, the chosen pair of adjacent actors have the maximum repetition count value ρ see Definition definition 1, associated to their inter-connected edge.

Definition 1. *If Z is a subset of actors in a connected, consistent SDF graph:*
 $\rho(Z) \equiv \gcd(\{q(A) \mid A \in Z\})$

APGAN candidates should respect the clustering rules that can be verified by applying a reachability matrix [3]. Then repeat the process until the end of the opportunities. Execution of hierarchical actor resulting from a clustering

operation is assumed to be atomic and is thus mapped and scheduled as a whole on a single core. In order to execute such cluster actor, it is thus necessary to compute a sequential schedule of all actor firings that belong to the cluster. APGAN algorithm also provides special clustering schedules that are nested looped schedules whose specificity is to make sequential the behavior of a cluster. APGAN clustering technique focuses on single-core optimization.

Four clustering techniques are presented in [9]. The first one empowers the user to select improper groups of actors, manual methods are tedious and may introduce deadlocks. The second one consists in clustering SDF subgraphs as long as possible. The third one is the Unique Repetition Count (URC) clustering technique developed in Sect. 3.1. The last one is an adaptation of Sarkar’s multiprocessor DAG scheduling heuristic [11] based on macro dataflow model in which the program is partitioned into tasks at compile time and the tasks are scheduled on processors at run time. All of these methods focus on the efficient reduction of the complexity of a graph without considering parallelism. The method introduced in this paper involves automatically generated architecture-adaptive parallel cluster instances.

3 SCAPE Method

The objective of the SCAPE method is to apply graph transformation to the SDF graph of an application, prior to its mapping and scheduling. To do so, clustering of actors within the input hierarchical SDF graph aims at reducing the complexity of the derived SrDAG used during mapping and scheduling. The SCAPE method aims at preserving the parallelism of the application so that it matches the parallel computing capabilities of the target architecture.

3.1 Design Space Exploration Optimisation

The SCAPE method is composed of three steps: configuration of the granularity, identification of particular patterns that will be the subject of clustering, and scaling up of the last clusters on the target architecture.

Configuration of the Granularity. The SCAPE method takes as input the PiSDF graph of n hierarchy levels that models the application and an integer value corresponding to the number n_c of hierarchy levels that the user wants to group coarsely. The output of the new method is a transformed graph with the RV \mathbf{q} associated with the actors located in the subgraphs on $n - n_c$ level reduced to the number of PE that compose the architecture. A graph on n levels will have $n + 2$ possible configurations of $n - n_c$ levels.

- Level 0 configuration: it is the state-of-the-art configuration where the entire graph is flattened before producing the SrDAG for scheduling.
- level $n + 1$ clustering configuration: it is grouping the entire graph into a single actor, thus resulting in a mono-core schedule.

- level 1 clustering configuration: it corresponds to generate groups on the bottom levels and reduce the RV \mathbf{q} associated with the actors located in the subgraphs on this level to the number of PE.
- level $l \mid l \in [2, n+1[$ clustering configuration: it corresponds to coarsely grouping bottom levels, generate groups on the just upper levels and reduce the RV \mathbf{q} of actors on this level to the number of PE Fig. 4.

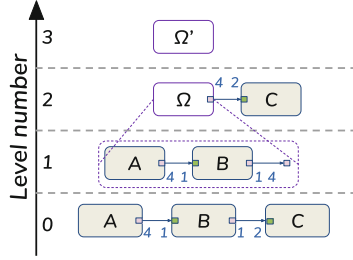


Fig. 4. Configuration based on clustering levels

Identification of Particular Patterns that will be the Subject of Clustering. The SCAPE method considers two patterns:

URC pattern: It's a sequence of at least two sequential actors with the same repetition count value ρ see Definition definition 1 without internal state. Such a pattern can be the object of a cluster if the consistency of the graph is preserved that can be verify by applying a reachability matrix.

Example 2. Considering the graph G shown in Fig. 5 which contains a sequence of actors B, C, D , and each FIFO connecting these actors presents the same repetition count value ρ , $\rho(A, B) = \gcd(1, 4) = 1$ and $\rho(\Omega, C) = \gcd(1, 2) = 1$. The method transforms the graph G by replacing this identified group with a hierarchical actor whose behavior is specified by a subgraph containing the identified group. This way the newly created hierarchical actor executes once per iteration and the elements it contains keep their initial execution number. Thus the SrDAG transformation of this piece of graph which would have resulted in $3 \times 8 = 24$ actors is presently 1 actor.

Single Repetition Vector (SRV) pattern: It's a single actor that does not belong to an URC candidate, with a RV \mathbf{q} greater than or equal to the number of PEs of the target architecture.

Example 3. We consider the graph G shown in Fig. 5 which contains an actor E with a RV \mathbf{q} equal to 16 and a target architecture which contains 4 cores. The method transforms the graph G by replacing this identified actor by a hierarchical actor whose behavior is specified by a subgraph containing the identified actor. This way the hierarchical actor executes once per iteration and the element it contains keeps its native execution number. Thus the SrDAG transformation of this piece of graph which would have resulted in 16 actors is presently 1 actor.

These two identified patterns will drastically reduce the size of the SrDAG and consequently make the application intrinsically sequential.

Scaling up of Cluster. The final step, called the *scaling*, consists of creating clusters of actors with a RV \mathbf{q} matching the target architecture. The scaling is done on the hierarchical actors located on the subgraphs at the level n_c input integer value. Level 0 and $n+2$ clustering are not subject to scaling. According to [8] to preserve the consistency of a graph G , on each FIFO f the rates of consumed and produced tokens *cons* and *prod* and the RV \mathbf{q} of the source and sink actors *src* and *snk* are linked by the equation:

$$\mathbf{q}(\text{src}(f)) \times \text{prod}(f) = \mathbf{q}(\text{snk}(f)) \times \text{cons}(f) \quad (1)$$

To calculate the scaling, the RV of the hierarchical actor $\mathbf{q}(h_a)$ shall be equal to the greatest common divisor of the RVs of the actors of the subgraph C flattened just above the number of PE n_{PE} .

$$\mathbf{q}(h_a) = \text{gcd}(\mathbf{q}(a \in C) \mid \mathbf{q}(h_a) \geq n_{PE}) \quad (2)$$

In case the hierarchical actor contains a FIFO with a number of delay D , special care must be taken when *scaling* the actor. In particular, if the hierarchical actor is directly connected to a delayed FIFO or indirectly via a special actor or an interface connected to a delayed FIFO. If one condition holds true then the calculation of the scaling is indexed on the delay value such as the rates of consumed tokens on the delayed FIFO $\text{cons}(f_{h_{a_d}})$ has to be less than or equal to the delay value D .

$$\mathbf{q}(h_a) = \text{gcd}(\mathbf{q}(a \in C) \mid \text{cons}(f_{h_{a_d}}) \leq D) \quad (3)$$

In order to keep the consistency of the graph, the rates of tokens consumed and produced on the input and output ports by the hierarchical actor $\text{in}(h_a)$ and $\text{out}(h_a)$, f for final and i for the initial value, are scaled as follow:

$$\begin{cases} \text{in}(h_a)_f = \text{in}(h_a)_i \times \mathbf{q}(h_a)_i / \mathbf{q}(h_a)_f \\ \text{out}(h_a)_f = \text{out}(h_a)_i \times \mathbf{q}(h_a)_i / \mathbf{q}(h_a)_f \end{cases} \quad (4)$$

Thus the actors from the subgraph are executed $\mathbf{q}(a \in C) / \mathbf{q}(h_a)$ times.

Example 4. We consider the graph G shown in Fig. 5 and a target architecture with 4 PEs. As the RV \mathbf{q} of the URC cluster is $\mathbf{q}(a \in \text{URC}) = 8$, then the scaling will be $\text{gcd}(8, 4) = 4$. Thus, the hierarchical actor URC executes 4 times and the subgraph elements twice per iteration. Respectively as the RV \mathbf{q} of the SRV cluster is $\mathbf{q}(a \in \text{SRV}) = 16$, then the scaling will be $\text{gcd}(16, 4) = 4$. Thus, the hierarchical actor SRV executes 4 times and the subgraph elements 4 times per iteration. From the input graph, the classic “flattening” approach obtains a size of the SrDAG of 50 actors, the “SCAPE” approach obtains a size of the SrDAG of 10 actors. The method reduces both the number of actors related to calculations and the number of special actors related to data transfers on the different instances. The complexity of the graph has been divided by 5, which considerably reduces the mapping and scheduling time of the tool without compromising the parallelism.

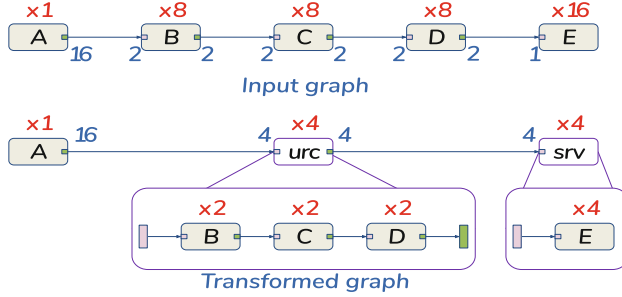


Fig. 5. SCAPE method

3.2 Code Generation

The code generated by the classic flattening approach in our tool [5] takes the form of a specific C file for each target PE. Every file contains first of all a part dedicated to the initialization of the application which includes the definition of the allocated buffers, actors and FIFOs initialization functions such as delay initialization. The second part of these files is a loop representing the thread containing the scheduled firing of actors. It is a function call implementing the behavior of the actor. Up until now, the tool did not support code generation for optimized actor groups. A plugin has been created to answer this new constraint. Thanks to this plugin, a cluster of actors is translated by nested function calls depending on whether the group contains other groups and the firing instances of the group elements are translated by “for” loops Fig. 6.

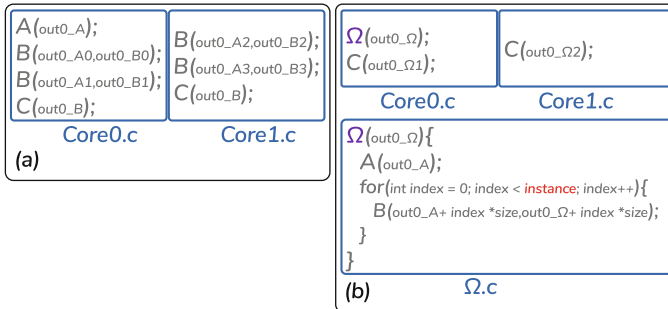


Fig. 6. Considering the graph from Fig. 4 and a two-core architecture (a) illustrates the code generation on clustering configuration with level 0, and (b) illustrates the code generation on clustering configuration on 2 levels

4 Experiments

The purpose of this section is to show that the proposed method offers a trade-off between reducing mapping and scheduling time, also called analysis time,

while preserving the latency of applications in comparison to the classic flattening method. The proposed method has been implemented in open source projects into Parallel and Real-time Embedded Executives Scheduling Method (PREESM) rapid prototyping framework. Compared to other mapping and scheduling frameworks, the absolute analysis time may seem high for DAGs with a few thousand actors. This time is due to the language used, Java, and heavy-weight implementation choices made by PREESM. Nevertheless, the comparison of the evolution of analysis times, which relates to their complexity, would remain valid with faster implementations of both the state-of-the-art corresponding to “Level 0” and the proposed technique. The experiments are performed on a desktop computer with an 8-core Intel i7-8665U processor and 31,2 GB of RAM.

4.1 Experimental Setup

Figure 7 presents the “analysis” and latency measured for the stereo application. The application has 2 levels of hierarchy, so there are 4 possible clustering configurations as explained in the section Sect. 3.1. Three image processing application use-cases such as *Stereo*, *Stabilization* and *Squeezenet* were used to conduct the experiments on architectures with a 1, 2, 4, 8 or 16 of homogeneous cores summarized in the Table 1. These models were chosen because they do not contain too many delays, which impends the scaling opportunities of the SCAPE method. These applications have between 2 and 3 levels of hierarchy. For each number of cores, only the result giving the best latency was kept, among all levels of the SCAPE method.

4.2 Analysis Time Evaluation

The experimental results depicted in Fig. 7 compare in red the state-of-the-art configuration and the different shades of blue for the different levels of clustering configuration up to 0. The analysis time curves are shown on the left side of Fig. 7. The curves representing the clustering configuration on different levels are between two extremes. The highest curve named “Level 0” represents the more complex graph with a time that increases with the number of cores due to the time that the mapping and scheduling algorithm takes to map, schedule and allocate memory to each of the elements of the SrDAG. The lowest curve: the fully cluster configuration remains constant and fast whatever the number of cores but no parallelism.

4.3 Latency Evaluation

The latency curves are shown on the right side of figure Fig. 7. There are still two extreme curves: the “level 0” curve whose complexity allows to distribute the actors on the different cores. That’s why the latency decreases with the number of cores. The level 3 clustering configuration, because it is sequential, has the longest latency and is architecture-independent. Thus the different clustering

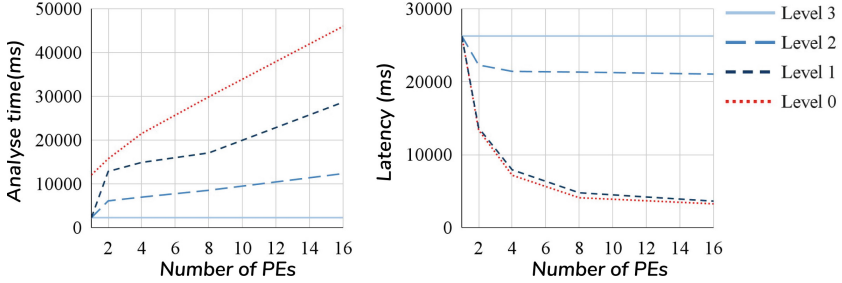


Fig. 7. Comparison of analysis time and latency between the classical flattening approach and three configurations of SCAPE method on *Stereo* application on several architectures (Color figure online)

configurations offer graphs of different levels of granularity and provide a tradeoff between the analysis time and the latency of an application.

Results depicted in Table 1 correspond to the ratio between the times obtained on the state-of-the-art configuration, “level 0”, and those obtained on the clustering configuration that offers the best compromise between analysis time and latency. A value greater than 1 is a speedup. The values obtained on the *Squeezenet* application on level 0 configuration are estimated with an exponential function. The process exceeded the RAM memory capacity of the used machine, due to the large number of actors in the SrDAG on the state-of-the-art configuration, and was unable to complete after 48 h. Hence the relevance of the method that allows to provide analysis and executable code even on very complex applications.

Table 1. Comparison of analysis time and latency between the classic flattening approach and best configurations of SCAPE method on three use-cases

Application	SDF	Level	SrDAG	Relative time	Number of PEs				
					1	2	4	8	16
Stereo	28	2	187	analysis	5.3	1.2	1.4	1.7	1.6
				execution	1.0	0.9	0.9	0.8	0.8
Stabilization	22	3	98	analysis	1.5	0.5	0.5	0.6	0.7
				execution	1.0	1.0	0.7	0.7	0.8
Squeezenet	98	3	5452	analysis*	203.5k	100.5	94.5	84.2	68.8
				execution*	1.0	1.0	1.0	1.0	1.0

*Estimated values

5 Conclusion

This paper presents a new method to reduce mapping and scheduling time while preserving the parallelism of SDF graphs. It consists in reducing the size of the














graph by clustering actors reproducing particular patterns and then reducing the firing instances of these clusters on the target architecture. The method allows the user to choose the potential expense of the produced schedule and reduce the analysis time accordingly. Experimental results show that for a significantly improved analysis time we obtain a slightly deteriorated latency of the generated code. In addition, the methods enable mapping and scheduling massively parallel applications which were too complex for state-of-the-art approaches. Potential directions for future work include identifying and clustering more complex patterns and automating the search for the optimal level of clustering, without needing to try all configurations.

References

1. Alok, G.: Architecture apocalypse dream architecture for deep learning inference and compute-versal ai core. *Embedded World* (2020)
2. Bhattacharyya, S., Murthy, P., Lee, E.: APGAN and RPMC: complementary heuristics for translating DSP block diagrams into efficient software implementations. *Des. Autom. Embedded Syst.* **2**, 33–60 (1997)
3. Bhattacharyya, S.S., Lee, E.A.: Scheduling synchronous dataflow graphs for efficient looping. *J. VLSI Signal Process. Syst.* **6**(3), 271–288 (1993)
4. Desnos, K., Heulot, J.: PiSDF: parameterized & interfaced synchronous dataflow for MPSoCs runtime reconfiguration. In: *1st Workshop on Methods and Tools for Dataflow Programming (METODO)*. ECSI, Madrid, Spain (2014)
5. Heulot, J., et al.: An experimental toolchain based on high-level dataflow models of computation for heterogeneous mpso. In: *Proceedings of the 2012 Conference on Design and Architectures for Signal and Image Processing*, pp. 1–2 (2012)
6. Klikpo, E.C., Khatib, J., Munier-Kordon, A.: Modeling multi-periodic simulink systems by synchronous dataflow graphs. In: *2016 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, pp. 1–10 (2016)
7. Lee, E., Ha, S.: Scheduling strategies for multiprocessor real-time DSP. In: *1989 IEEE Global Telecommunications Conference and Exhibition 'Communications Technology for the 1990s and Beyond'*, vol 2, pp. 1279–1283 (1989)
8. Lee, E.A., Messerschmitt, D.G.: Static scheduling of synchronous data flow programs for digital signal processing. *IEEE Trans. Comput.* **C-36**(1), 24–35 (1987)
9. Pino, J., Bhattacharyya, S., Lee, E.: A hierarchical multiprocessor scheduling system for DSP applications. In: *Conference Record of The 29th Asilomar Conference on Signals, Systems and Computers*, vol. 1, pp. 122–126 (1995)
10. Pino, J., Bhattacharyya, S., Lee, E.: A hierarchical multiprocessor scheduling framework for synchronous dataflow graphs (1995)
11. Sarkar, V.: Partitioning and scheduling parallel programs for execution on multiprocessors (1987)
12. Sih, G., Lee, E.: Scheduling to account for interprocessor communication within interconnection-constrained processor networks, pp. 9–16 (1990)
13. Zhou, Z., Desnos, K., Pelcat, M., Nezan, J., Plishker, W., Bhattacharyya, S.: Scheduling of parallelized synchronous dataflow actors (2013)



Deep Recurrent Neural Network Performing Spectral Recurrence on Hyperspectral Images for Brain Tissue Classification

Pedro L. Cebrián¹ , Alberto Martín-Pérez¹ , Manuel Villa¹ ,
Jaime Sancho¹ , Gonzalo Rosa¹ , Guillermo Vazquez¹ ,
Pallab Sutradhar¹ , Alejandro Martínez de Ternerero¹ ,
Miguel Chavarrías¹  , Alfonso Lagares² , Eduardo Juárez¹ ,
and César Sanz¹ 

¹ Universidad Politécnica de Madrid (UPM), 28040 Madrid, Spain
{pl.cebrian,a.martinp,manuel.villa.romero,jaime.sancho,
gonzalo.rosa.olmeda,guillermo.vazquez.valle,pallab.sutradhar,
a.mruiz,miguel.chavarrias,eduardo.juarez,cesar.sanz}@upm.es

² Instituto de Investigación Sanitaria Hospital 12 de Octubre (imas12),
28041 Madrid, Spain
alfonso.lagares@salud.madrid.org

Abstract. Hyperspectral imaging approaches have proven its effectiveness in the medical field for characterizing brain tissues. Furthermore, these techniques in conjunction with machine learning (ML) algorithms has shown to be a useful tool for tumor detection in order to assist neurosurgeons in operations. In this report, it is proposed a novel deep recurrent neural network (DRNN) performing spectral recurrence with the hyperspectral bands to increase precision in tissue classification. In addition, this research present a comparison between the optimized models of the followings ML algorithms: support vector machine, random forest (RF) and the DRNN. All of them were trained by following an hyperparameter optimization process. As a result, DRNN improve brain tissue predictions in terms of the area under the receiver operating characteristic objective test metric (by 1.39% over SVM and 1.91% over RF) whereas RF classification maps illustrate truthfully the distribution of different tissue regions.

Keywords: HSI · RNN · hyperparameter · optimization · brain tumor

1 Introduction

The use of different imaging-based diagnosis techniques in medicine is nowadays widespread and provides an important support in almost all areas of healthcare

This work was supported by the Spanish Government through TALENT-HIPSTER project (PID2020-116417RB-C41).

practice. The treatment of brain tumors, far from being an exception, represents a challenge where techniques such as computed tomography, positive emission tomography or magnetic resonance imaging (MRI) are daily used [1]. Moreover, these technologies may be combined with machine learning (ML) algorithms to extract from images hidden information and better supporting technicians in diagnosis and medical practice. Recently, Hyperspectral (HS) imaging (HSI) is gaining presence in medical field due to its non-invasive and non-ionizing features. This emerging technique is related to spectroscopy [2] and allows to obtain the spectral features of a captured material. The spectral information is organised in three-dimensional cubes containing the spatial and spectral information of the region of interest. However, due to the inherent complexity of living tissue, ML techniques need to be applied to draw conclusions from the volume of data generated by hyperspectral cameras. In this sense, some works have been carried out involving different ML algorithms such as support vector machine (SVM) and random forest (RF) based classifiers. Both algorithms are compared for brain tissue classification using HS images in [3].

On the basis of neural networks (NN), recurrent neural network (RNN) architecture is useful for extracting temporal data dependencies (i.e. RNN is used to extract features from MRI sequences for brain tumors detection [4]). In this work, it is proposed to use a RNN performing spectral recurrence in order to profit on correlation between HS frequency bands when classifying brain tumor tissues during neurosurgical procedures. To this end, a new deep recurrent neural network (DRNN) architecture has been i) tailored using as reference the state of the art of RNN in remote sensing [5], and ii) optimised using a hyperparameter optimization method. In addition, SVM and RF models from [6] have been re-trained following the same optimization methodology that the DRNN model. All in all, the obtained results show the behaviour of each model performing brain tissue classification task.

According to what the authors know, there is previous research assessing a RNN architecture in tumor analysis (head and neck regions) by medical HSI [7]. However, this work evaluates an optimized RNN model in brain tissues classification from HS images by performing recurrence between spectral information.

2 Background

2.1 Hyperspectral Imaging

HS images consist of hundreds (typically between 100 and 250) narrow wavelengths bands (around 3–10 nm bandwidth) that collect a range of continuous information from the electromagnetic spectrum energy reflected for each pixel of a capture. This information is called spectral signature and is used to difference between materials on the image. Currently, HS images are applied in several fields such as food quality evaluation (i.e. minced meat classification [8]) or cancer detection in medicine (i.e. gastric cancer diagnosis [9]).

2.2 Machine Learning Algorithms

ML is a branch of artificial intelligence field that aims to emulate human intelligence capacity in computer systems by learning through experience. ML trained models are capable to detect patterns and make predictions in an analytical and autonomous way without being explicitly programmed to do so.

There are four sub-types of ML algorithms depending on the learning method: supervised learning, unsupervised learning, semi-supervised learning and reinforcement learning. Also, within the supervised learning algorithms there are classification and regression models according to the task to perform. In this work, it is compared three different ML models trained on a supervised way for a classification problem:

- SVM [10] works by finding the optimal hyperplane in a n-dimensional space to separate labelled data points of the training set into different classes. The number of space dimensions refers n data features.
- RF [11] consist of a parallel set of individual decision trees that operates as an ensemble to obtain an outcome based on the majority prediction of them. Input training data set is statistically split as bootstrap samples for each decision tree.
- RNN is another type of NN based on a feed-forward NN [12] but with recurrent neurons on hidden layers that feedback themselves sequentially. It shares orderly the training weights information between sequence steps or time steps allowing giving a memory mechanisms to the NN. There are three different recurrent units for the neurons: vanilla recurrent unit, long short-term memory (LSTM), and gated recurrent unit (GRU). While the vanilla unit is the oldest and simplest model, LSTM and GRU are sophisticated units that avoid vanish and exploding gradient [13].

2.3 Hyperparameter Optimization

ML models have neuron weights (also called parameters) that are updated during training process depending on the input data. However, ML hyperparameters are the ones that can be configured before the training process and their values influence the weights setting during training. Typically, hyperparameters values are configured manually, but it is tedious to establish them on larger hyperparameter sets and complex ML algorithms. Also, it does not ensures a high ML model performance. Alternatively, there is an automatic method for hyperparameters assignation. This method requires a search space with hyperparameters sets, an objective score function with a direction to achieve (minimize or maximize) and an algorithm to select the specific set of values for the next configuration trial. The aim of hyperparameter optimization in ML is to find a configuration of them that reaches the best performance of the model on a validation dataset in terms of the objective metric established (function and direction). There are different algorithms such as grid search (GS) or random search (RS) to perform the hyperparameter selection. The basis of this work is related to Bayesian

optimization (BO) algorithm. Unlike GS and RS, BO evaluates the selection by mapping hyperparameters to a probability of a score based on previous results of the function specified. This model is called surrogate, and works iterative approaching on the most promising hyperparameters values by paying attention to past evaluations in order to keep searching around this regions. Specifically, the sequential model-based optimization (SMBO) with tree-structured Parzen estimators (TPE) surrogate model [14] have been used in this work. This algorithm consist on running hyperparameter selection trials sequentially, employing Bayesian reasoning to update the surrogate model in order to select better hyperparameters each time. According to TPE surrogate function, Bayes rule is the methodology to compose the surrogate model in SMBO. A similar approach was used by Martín-Pérez et al. for the optimization of ML models (SVM and RF) for brain tissue classification, [6] comparing GS, RS and BO methods.

3 Methodology

In this paper, it is proposed a DRNN model tailored to the specific data described in Sect. 3.1 in order to improve previous results of brain tissue classification [3]. Concretely, the network architecture is presented in Sect. 3.2. Then, it is explained the training process with the hyperparameters optimization methodology in Sect. 3.3. In addition, SVM and RF algorithms described in [6] are used for the experiments suggested in Sect. 4 to compare the performance of the different ML models (DRNN, SVM and RF) in brain tissue classification.

3.1 Input Data

DRNN, SVM and RF models have been trained to classify brain tissue pixels between 4 types (healthy, tumor, blood and meninges) by using data previously labeled by neurosurgeons. These data are a collection of human brain HS images pre-processed from different patients and operations. Specifically, HS images captured are acquired in *in-vivo* surgical interventions by employing the camera Ximea Snapshot MQ022HG-IM-SM5X5-NIR 1st generation which consist of information with 25 spectral bands captured in the 665–960 nm range. The description of these images and the pre-process methods are detailed in [3]. For each pre-processed HS brain tissue image there is a ground truth (GT) map that contains labeled pixels corresponding to neurosurgeons observations (see left side of the Fig. 1). Thus, this GT map is used to check the network output comparing the outcome labels with the real ones during training. Specifically, these labels correspond to the 4 types of tissues mentioned, taking into account that 1) blood tissue type involves veins and arteries and 2) that meninges consist of three different membranes that cover and protect the brain. In this work, data is handled as in [15] to extract patches from a HS image. As it is illustrated in Fig. 1 the process consist on: 1) to generate spatial squares ($patch_size \times patch_size$) around each labeled pixel on the GT map, and 2) to create patches extracting the values of the 25 frequency bands from the HS image taking into account previous

squares as a spatial reference. Finally, patches are reshaped on a single-spatial dimension element (vPatch: vector patch) with the values of the 25 frequency bands on the spectral dimension ($\lambda_1, \lambda_2, \dots, \lambda_{24},$ and λ_{25}). Thus, as it is illustrated in Fig. 2, spatial dimensions of the patch are stretched to obtain vectors ($patch_size \times patch_size$) for each frequency band in order to work with a RNN.

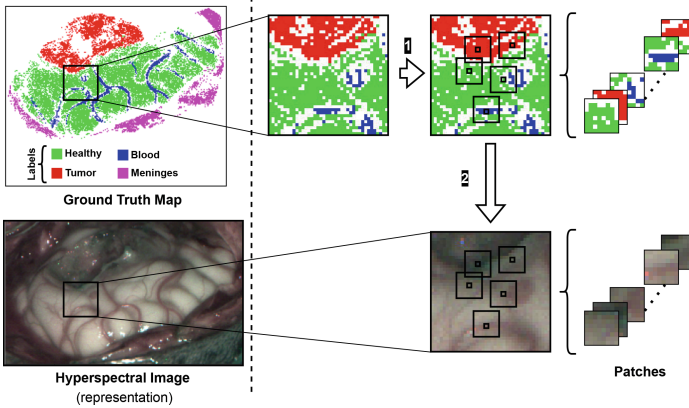


Fig. 1. Diagram of patches extraction from a HS image. On the right side of the dashed line there are a GT map and the HS image of one patient. On the other side 1) squares selection and 2) patch extraction are represented.

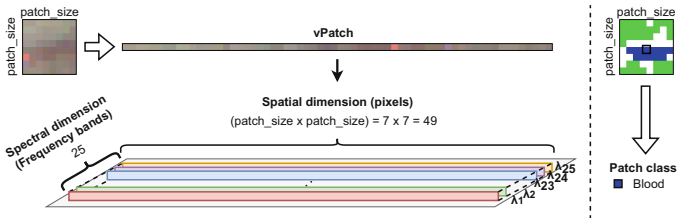


Fig. 2. Diagram with the generation of the network input vector and the target output label. The vPatch composition is represented on the right side of the dashed line whereas the patch class target is illustrated on the other side.

3.2 Deep Recurrent Neural Network Architecture

On the basis of RNN, proposed network architecture use the time steps dependence in order to take advantage of the frequency bands correlation on HS images. This frequency bands are considered as a sequence being used on time steps one by one in order to perform a band-by-band processing [5]. Therefore, DRNN is implemented with 25 time steps to process 25-frequency-bands data. It also include a symmetric transfer of learning weights information between

bands by sharing hidden states bidirectionally from the first hyperspectral band to the last one ($h_0, h_1, \dots, h_{24}, h_{25}$) and the other way around (from h'_0 to h'_{25}). The architecture of the network is outlined on Fig. 3 and consist on two recurrent hidden layers composed by 208 LSTM recurrent cells to learn deeply about data features and avoid learning problems such as vanishing or exploding gradient. Also, a dropout rate of 0.6 is used in each hidden layer to reduce over-fitting during training. In addition, any recurrent cell use an hyperbolic tangent as activation function. Then, as a many-to-many (n-to-n, $n = 25$) RNN topology, DRNN architecture obtain an outcome vector $[nb_hyperspectral_bands, nb_rnn_cells \cdot 2(bidirectional)] = [25, 208 \cdot 2]$ gathering every time step output. Finally, a fully-connected layer with 4 neurons and a softmax activation function performs the multi-class classification with the previous resulting vector. This layer obtains a single [4,1] vector with the probabilities of a vPatch for being each brain tissue type: healthy, tumor, blood and meninges.

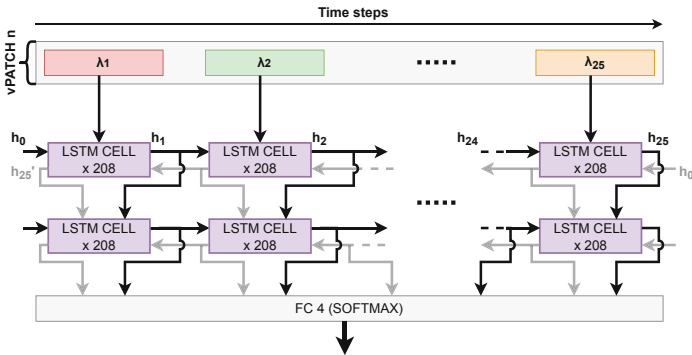


Fig. 3. Diagram of the DRNN architecture.

The input of the network is the HS vector (vPatch) composed by a certain number of pixels ($patch_size \times patch_size = 7 \times 7 = 49$) and the information about the 25 spectral bands. Concretely, $patch_size = 7$ is selected based on experiments in [3] to employ an enough amount of neighbouring pixels that improve classification results without decreasing classification map resolution. On top of Fig. 3 it is represented the input of the network, where each vector (with the values of one specific spectral band λ_i , $i = 1, 2, 3, \dots, 24, 25$) goes into one time step. Then, at the output of the DRNN there is a last layer vector [4,1] of outcome scores (before the softmax activation function) whose major probability type is compared with the labeled tissue type for the input vPatch. Specifically, cross-entropy loss function is used to compare the outcome predicted label with the reference in order to train the network parameters (weights) by minimizing this loss result on each training step (epoch). Related with the training process of the ML model, 15 HS images which have hundreds of labeled pixels are involved. For this reason there is a large amount of patches to be handled. Thus, batches

are used to distribute data on the training process in order to control the power consumption and memory resources. In particular, a *batch_size* of 40 is applied so each iteration of the training process involves 40 patches through the network.

3.3 Training Process

The training process of the DRNN model consist on performing hyperparameter optimization applying SMBO with TPE algorithm. Thus, the optimal set of hyperparameters values could be obtained according to the classification task performance. Additionally, this technique is combined with a cross validation (CV) procedure to reinforce the accuracy of the hyperparameter set results from the data variability. In more detail, the training process configuration is based on the methodology presented in [6]. Accordingly, it is implemented in this work by replacing the *StratifiedShuffleSplit* function for a Kfold CV (with $K = 5$ folds as in [3]) in order not to repeat data in different splits. Also, this work involves alternative implementations for stratifying and randomly shuffling data samples (patches). Specifically, it works by stacking different data samples by type and randomly shuffling them with a particular seed. According to the hyperparameters involved in the optimization process, there are some of them related to training hyperparameters (the number of epochs, the learning rate, the optimizer and the size of each batch), whereas others are part of the RNN architecture (the type of the recurrent cells and the number of recurrent neurons/cells for each layer). The distributions of values for each hyperparameter evaluated are presented in Table 1. Following the presented methodology, the proposed solution performs a 100-trials optimization minimizing the average cross-entropy loss function of the five partial results from the 5-fold CV. Furthermore, data sampled used in this training process (with the hyperparameter optimization) consist of the 80% randomised-stratified labeled patches from 15 HS images. Then, the other 20% of the labeled samples is used with the trained DRNN model in order to test/validate its classification results. Finally, as a result of the hyperparameter optimization, best values selected for the hyperparameters are expressed in the Result column of the Table 1.

Table 1. Distribution values arrangement for the hyperparameter optimization study and its corresponding results.

Hyperparameter	Distribution	Result
<i>Epochs</i>	[160, 320] in integer steps of 10	290
<i>Learning_rate</i>	[1e-5, 1e-2] in log steps	5.975e-4
<i>Batch_size</i>	[32, 128] in integer steps of 8	40
<i>Optimizer_name</i>	[Adam, Adagrad, AdamW]	AdamW
<i>Hidden_size</i>	[64, 256] in integer steps of 12	208
<i>RNN_model_type</i>	[Basic RNN (vanilla), LSTM, GRU]	LSTM

4 Experiments and Results

Experiments accomplished in this work consist on performing training process based on hyperparameter optimization with data from a specific test bench of 16 patients, as it is detailed in Sect. 4.1. SVM, RF (from [6]) and DRNN trained model are evaluated on brain tissue classification task. Specifically, models are compared with each other according to the area under the receiver operating characteristic curve (ROCAUC) metric presented in Sect. 4.2. Finally, the analysis of the experiments results takes place in Sect. 4.3.

4.1 Test Bench

As it is described in Sect. 3.1, data from real brain surgery captures are used for training and testing the ML models. Also, these data are employed to perform classification predictions with trained models in order to evaluate its performance. Concretely, it is selected a set of 16 captures of patients who suffer from glioblastoma with non-mutated IDH gene. Here below, the identifiers and also the pixel distribution of these captures are presented in Table 2. These captures are divided into two groups: group A and group B. The first one includes the 15 captures used in training process whereas the second one only include one capture used to perform a new patient classification process with the trained model. In fact, group A pixels are divided into two data splits. The first one is composed by 80% labeled pixels whereas the other set only has the remaining 20%. Both groups have been composed by stratifying and randomizing the distribution of the labeled pixels data samples due to the unbalanced classes arrangement. Then, as it is described in Sect. 3.3, the 80% dataset is employed for training the different models (SVM, RF, DRNN) with pairs of patch-label. After that, the other split is applied for testing models. Finally, each individual capture is used to perform a classification of all its pixels with the trained model. Some labeled pixels on Group A captures take part into the training process. Thus, ID71C02 capture has been separated in order to classify a capture whose pixels are new to the trained model. This second type of experiment (B) represents the behaviour of a trained model on a real classification as it could be working on a in-vivo brain surgery with a new patient.

4.2 Metrics Evaluated

In order to compare different results on brain tissue HS images classification it is employed the ROCAUC, that is a threshold-invariant and scale-invariant metric used to obtain a reliable global measurement of model performance. So it could be useful in a objectively comparison between different ML models. Here below, it is included receiver operating characteristic (ROC) expression in Eq. 1 and ROCAUC value formula in Eq. 2. Additionally, there are two expressions added for partial values of the ROC equation: true positive rate (TPR) in Eq. 3, and false positive rate (FPR) in Eq. 4. It is also notable to mention that TP, TN, FP and FN values of this expressions are respectively: true positives, true negatives,

false positives and false negatives predicted values obtained after classifying with a ML model.

Table 2. Total number of labeled pixels by patient capture and its distribution into the different types of brain tissues evaluated: healthy, tumor, blood and meninges.

Patient ID	healthy	tumor	blood	meninges	Total
Group A					
ID18C09	648	1587	93	369	2697
ID25C02	801	206	105	94	1206
ID29C02	3752	64	109	1599	5524
ID30C02	2587	2737	868	1366	7558
ID34C02	1186	1464	357	780	3787
ID38C02	1740	487	304	825	3356
ID47C02	174	160	112	567	1013
ID47C08	715	182	162	129	1188
ID50C05	410	1282	759	628	3079
ID65C01	1759	1039	111	1426	4335
ID67C01	2651	579	388	422	4040
ID70C02	1486	769	190	551	2996
ID72C02	512	760	148	1430	2850
ID75C05	1588	400	407	-	2395
ID84C02	1898	837	-	671	3406
Group B					
ID71C02	3044	174	2144	1763	7125

$$ROC = \left\{ \left(FPR(t), TPR(t) \right) \right\}_{t=0}^1 \quad (1) \quad ROCAUC = \int_{t=0}^1 ROC(t) dt \quad (2)$$

$$TPR = \frac{TP}{TP + FN} \quad (3) \quad FPR = \frac{FP}{FP + TN} \quad (4)$$

4.3 Analysis

In this work it is presented a comparison between evaluated ML optimized models (SVM, RF, DRNN) in terms of: 1) ROCAUC metrics on validation process (with the test samples of the 20% split) and ROCAUC metric on a real classification process (with labeled pixels from capture ID71C02), both expressed in Table 3, 2) classification maps for the ID67C01 capture that are shown in Fig. 4 and 3) classification maps for the ID71C02 capture presented in Fig. 5.

According to ROCAUC metrics in Table 3 calculated for the 20% test split, DRNN optimized model demonstrate a higher yield than the other ones in terms of classification. Concretely, DRNN improves ROCAUC test metric averaging all classes by 1.39% over SVM and 1.91% over RF. In addition, DRNN increase ROCAUC metric results of the ID71C02 labeled pixels classification for the healthy, blood and meninges tissue types. Specifically, as it is shown in the column “ID71C02 labeled pixels” of the Table 3, DRNN has an improvement rate of: 1) 2.22% and 0.18% for healthy pixels classification compared to SVM and RF models respectively, 2) 10.71% and 3.91% for blood pixels classification compared to SVM and RF models respectively, and 3) 8.36% and 7.39% for meninges pixels classification compared to SVM and RF models respectively. However, RF ROCAUC value for tumor classification improves by 6.59% the value of the DRNN model.

Table 3. ROCAUC % outcomes obtained for each ML model (SVM, RF, and DRNN) with: the (20%) test samples dataset, and labeled pixels from capture ID71C02.

Class	20% Test dataset			ID71C02 labeled pixels		
	SVM	RF	DRNN	SVM	RF	DRNN
Healthy	99.01	98.43	99.96	97.16	99.20	99.38
Tumor	98.01	97.07	99.96	62.22	85.95	79.36
Blood	98.19	98.02	99.96	82.18	88.98	92.89
Meninges	99.08	98.71	99.99	88.88	89.85	97.24

Regarding graphical analysis, there are classification maps from one capture involved in training process (ID67C01) and a capture which is not involved (ID71C02) presented in this work. For the outcomes of the first experiment, it is illustrate in Fig. 4 that SVM and RF optimized models segment properly blood tissue regions concerning to the GT. However, DRNN model classification fits better according to the tumor detection due to the pixels noise in SVM and RF model outcomes. This noise is the result of incorrectly predicted tumor pixels.

Then, results from second experiment (see Fig. 5) expose that DRNN model classification is capable to properly detect different regions according to GT. However, tumor tissue predictions include some small areas incorrectly predicted close to the real tumor region.

It is also impressive to mention that despite pixels noise in RF classification map, this optimized model accurately detect the main tumor region concerning to the GT. Thus, an adequate representation regarding the GT could be obtained by applying some filters to this classification map. Therefore, it is possible tumor segmentation with this RF optimized model.

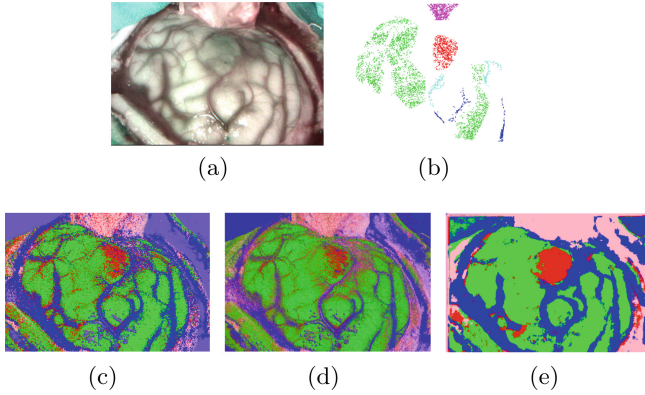


Fig. 4. ID67C01 patient capture where (a) is the pseudo RGB of the HS capture, (b) is the ground-truth map and (c) to (e) are the probability classification maps using SVM, RF and the proposed DRNN model respectively. Healthy, tumor, blood and meninges colors are green, red, blue and pink, respectively. (Color figure online)

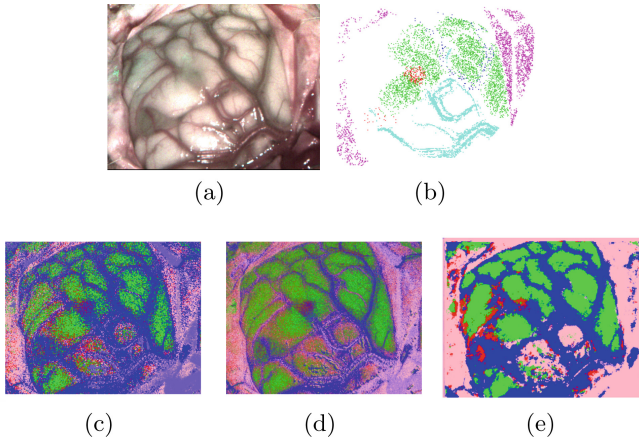


Fig. 5. ID71C02 patient capture where (a) is the pseudo RGB of the HS capture, (b) is the ground-truth map and (c) to (e) are the probability classification maps using SVM, RF and the proposed DRNN model respectively. Healthy, tumor, blood and meninges colors are green, red, blue and pink, respectively. (Color figure online)

5 Conclusions and Future Work

In this work, it is presented a new DRNN architecture that is potentially useful in HSI classification for brain tissues. It increase the real-case classification performance (capture ID71C02) regarding SVM or RF. Specifically, DRNN improves ROCAUC metric by 0.18%, 3.91% and 7.39% over RF for healthy, blood and meninges classes respectively. However, it is impressive to take into account that

this classification task requires not only objective metrics, but also graphics representations in order to resemble outcomes to their real use case. That is why it could be interesting to consider RF optimized model as a positive candidate to perform the real classification by applying some filters to its classification map.

Finally, as a possible future working lines it is suggested: 1) to employ this DRNN architecture combined with convolutional neural networks in order to extract spatial-spectral features. Thus, it could be obtained a ML model capable to increase previous classification results. And 2) to create a ML model based on RF optimized and arrange a digital filtering pipeline in order to reduce pixel noise and obtain a classification map similar to the GT. In both lines, it could be interesting to use additional patient captures to provide extra data for training the ML model.

References

1. Kraus, G.E., et al.: A technique utilizing positron emission tomography and magnetic resonance/computed tomography image fusion to aid in surgical navigation and tumor volume determination. *J. Image Guid. Surg.*, vol. 1, pp. 300–307 (1995). [https://doi.org/10.1002/\(SICI\)1522-712X\(1995\)1:6<300::AID-IGS2>3.0.CO;2-E](https://doi.org/10.1002/(SICI)1522-712X(1995)1:6<300::AID-IGS2>3.0.CO;2-E)
2. ElMasry, G., Sun, D.W.: Principles of hyperspectral imaging technology. In: *Hyperspectral Imaging for Food Quality Analysis and Control*. Academic Press, p. 3–43 (2010). <https://doi.org/10.1016/B978-0-12-374753-2.10001-2>
3. Urbanos, G., et al.: Supervised machine learning methods and hyperspectral imaging techniques jointly applied for brain cancer classification. *Sensors* **21**(11), 3827 (2021). <https://doi.org/10.3390/s21113827>
4. Zhou, Y., et al.: Holistic brain tumor screening and classification based on DenseNet and recurrent neural network. In: Crimi, A., Bakas, S., Kuijff, H., Keyvan, F., Reyes, M., van Walsum, T. (eds.) *BrainLes 2018*. LNCS, vol. 11383, pp. 208–217. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-11723-8_21
5. Mou, L., Ghamisi, P., Zhu, X.X.: deep recurrent neural networks for hyperspectral image classification. *IEEE Trans. Geosci. Remote Sens.* **55**(7), 3639–3655 . <https://doi.org/10.1109/TGRS.2016.2636241>
6. Martín-Pérez, A., et al.: Hyperparameter optimization for brain tumor classification with hyperspectral images. In: *2022 25th Euromicro Conference on Digital System Design (DSD)* (2022). <https://doi.org/10.1109/DSD57027.2022.00117>
7. Bengs, M., et al.: Spectral-spatial recurrent-convolutional networks for *In-Vivo* hyperspectral tumor type classification. In: Martel, A.L., et al. (eds.) *MICCAI 2020*. LNCS, vol. 12263, pp. 690–699. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-59716-0_66
8. Ayaz, H., et al.: Hyperspectral imaging for minced meat classification using nonlinear deep features. *Appl. Sci.* **10**, 7783 (2020). <https://doi.org/10.3390/app10217783>
9. Knospe, L., et al.: New intraoperative imaging tools and image-guided surgery in gastric cancer surgery. *Diagnostics* **12**, 507 (2022). <https://doi.org/10.3390/diagnostics12020507>
10. Wang, L.: *Support Vector Machines: Theory and Applications*. Springer Science & Business Media, Auckland, vol. 177 (2005)
11. Breiman, L.: Random forests. *Mach. Learn.* **45**, 5–32 (2001). <https://doi.org/10.1023/A:1010933404324>

12. Svozil, D., Kvasnicka, V., Pospichal, J.: Introduction to multi-layer feed-forward neural networks. *Chemometri. Intell. Lab. Syst.* **39**(1), 43–62 (1997). [https://doi.org/10.1016/S0169-7439\(97\)00061-0](https://doi.org/10.1016/S0169-7439(97)00061-0)
13. Pascanu, R., et al.: On the difficulty of training recurrent neural networks. *Proceedings of the 30th International Conference on Machine Learning*, in *Proceedings of Machine Learning Research*, vol. 28, no. 3, pp. 1310–1318 (2013). <https://doi.org/10.48550/arXiv.1211.5063>. <https://proceedings.mlr.press/v28/pascanu13.html>
14. Bergstra, J., et al.: Algorithms for hyperparameter optimization. In: *Advances in Neural Information Processing Systems*, vol. 24 (2011). https://proceedings.neurips.cc/paper_files/paper/2011/file/86e8f7ab32cfd12577bc2619bc635690-Paper.pdf
15. Xu, Y., et al.: Hyperspectral image classification via a random patches network. *ISPRS J. Photogrammetry Remote Sens.* **142**, 344–357 (2018). <https://doi.org/10.1016/j.isprsjprs.2018.05.014>



Brain Blood Vessel Segmentation in Hyperspectral Images Through Linear Operators

Guillermo Vazquez¹, Manuel Villa¹, Alberto Martín-Pérez¹,
Jaime Sancho¹, Gonzalo Rosa¹, Pedro L. Cebrián¹, Pallab Sutradhar¹,
Alejandro Martínez de Ternero¹, Miguel Chavarrías¹,
Alfonso Lagares², Eduardo Juárez¹, and César Sanz¹

¹ Universidad Politécnica de Madrid, Madrid, Spain

{guillermo.vazquez.valle,manuel.villa.romero,a.martinp,jaime.sancho,
gonzalo.rosa.olmeda,pl.cebrian,pallab.sutradhar,a.mruiz,
miguel.chavarrias,eduardo.juarez,cesar.sanz}@upm.es

² Instituto de Investigación Sanitaria Hospital 12 de Octubre, Madrid, Spain
alfonso.lagares@salud.madrid.org

Abstract. Tissue classification tasks that rely on multidimensional data, such as spectral information, sometimes face issues related to the nature of their own characteristics when different biological components share similar spectrum. In a situation of in-vivo brain tumor location during a surgical operation, especially when applying machine learning techniques, relying solely on the spectral information of each sample may not be enough to provide a correct identification of all the tissues involved. In order to overcome this problem, in this work we propose to reduce conflicting classification pixels, i.e. vascular versus tumor tissues. To do so, morphological operators can supply support to a pixel-wise classification by exploiting the spatial characteristics present in vascular tissue. Hence, we have evaluated the suitability of linear operators for brain vessels segmentation in a context of hyperspectral video classification. The parameters of the operator along with the selection of the most suitable spectral band to process were chosen via optimization of the amount of vascular pixels detected and error metrics. The segmentation algorithm was implemented for both CPU and GPU platforms achieving a performance compatible with real-time classification purposes on the last one. Objective results show an average segmentation of the 68% of the vein and arteries present in the ground truth with less than a 10% of error selecting pixels from other tissues of interest such as healthy brain and tumor.

Keywords: Hyperspectral · segmentation · brain vessels · GPU

This work was supported by the Spanish Government through TALENT-HIPSTER project (PID2020-116417RB-C41).

© The Author(s), under exclusive license to Springer Nature Switzerland AG 2023
M. Chavarrías and A. Rodríguez (Eds.): DASIP 2023, LNCS 13879, pp. 28–39, 2023.
https://doi.org/10.1007/978-3-031-29970-4_3

1 Introduction

Non-invasive diagnosis through different digital imaging methods have accumulated a long history in which their integration into medicine has made the detection and treatment of certain diseases easier. In this field, hyperspectral imaging (HSI) has proven its usefulness in the optical characterization of the tissues [1, 2]. Through the spectral sampling of the reflectance captured from the scene, each biological component can provide what is called their spectral signature: a particular way each tissue disperses the incident light across the spectrum. This source of information, when combined with machine learning algorithms, can serve as a classification tool for certain pathologies such as tumor formations [3]. The problem comes when the spectral signatures are very similar between tissues, causing the classifier to mistake them. The consequence of this issue can be observed in [3], where the classification maps show a tendency to mark certain regions of the veins and the arteries on the surface of the brain as tumor despite their morphology being completely different. The ultimate goal of this classifier is to assist the surgeon performing the resection of the tumor by indicating its limits through a classification map. Therefore, any source of tumor false positives must be suppressed.

The mapping of the brain blood vessels through non-invasive techniques supposes a useful mechanism for the diagnosis of vascular pathologies and also assistance in surgical interventions. Normally, this mapping process is performed out from magnetic resonance images (MRI) or time of flight magnetic resonance angiography (ToF-MRA) where a series of images from transversal sections con-forms a volumetric representation of the head. The MRI and MRA images are capable of providing high spatial resolution of the internal structures of the brain. The existing literature that deals with the blood vessel segmentation matter from this kind of imaging is abundant: whereas deep learning-based techniques have gained popularity [4] in the last decade due to its prominent results [5], more traditional approaches based on morphological operations [6] are still capable of providing sufficient accuracy compared to nowadays standards. However, in situations where it is required to perform a segmentation of the cortical blood vessels of the brain through an open craniotomy, the images provided by an MRI scan become harder to exploit. This is because of the shifts the brain suffers during the surgical procedure causing the MRI to be difficult to match with the image from an external camera capturing the brain cortex. The set of techniques that relies only on external camera captures for the segmentation of vascular tissue is more scarce than those using the MRI and MRA imaging. Most of the literature focuses on ophthalmology applications for identifying the blood vessel structures [7] but when it comes to a brain surgery context, one of the few examples can be found in Wu et al. [8] where deep learning techniques are used to carry on the segmentation of the vessels of the mouse cerebral cortex. Also, in Fabelo et al. [9] this problem is addressed by classifying brain vascular tissue as a stage of a brain tumour detector. The use of neural networks for segmentation, such as the U-Net [10], has a particular aspect related to the training of the net: it requires a dense ground truth that has all the vascular elements to

be segmented labeled. If only a sparse ground truth is available, reconstruction processes like [11] can provide adequate training of the network, but they are only effective when there are certain gaps in the ground truth.

Some of the latest camera models equipped with snapshot sensors can capture hyperspectral video (HSV) bringing the opportunity and the challenge of performing a real-time classification over the captured sequence [12]. In this work we propose an efficient implementation of morphological operators based on GPU platforms. The goal is to perform a real-time segmentation of the brain vascular structures captured in a hyperspectral video from an in-vivo surgical intervention. This segmentation is intended to refine a classification map obtained from a classifier based on support vector machines (SVM) trained to detect brain tumor tissue. Hence, the importance of the efficiency of this correcting stage.

2 Background

The work described in this paper rests on two main basis:

1. Line operators: the work introduced by E. Ricci and R. Perfetti in [7] can be considered as the core of the proposed algorithm. Along with [6], it serves as an example of the suitability of this kind of operator for detecting blood vessels in an RGB image. Although in this paper the work material consists on hyperspectral images, they are processed as if they were regular captures. In [7], through a series of linear filters oriented by a constant increasing angle of 15° , the detector processed the inverted green channel of a non-mydratic retinal image obtained from the DRIVE [13] and STARE datasets [14]. The aim of the detector is to capture those vessels aligned with any of the linear filters to mark them in the output image. This linear detection is combined with an SVM to perform a binary classification. According to the results presented in the tables III and IV of the section IV in [7], the simple linear operator only performs a 0,8% less in the area under the curve of the ROC curve worse than the linear detector and SVM combined, at worst over the STARE dataset.
2. GPU data processing: medical image processing has experienced an important step forward thanks to the usage of GPU acceleration [15], either to shorten the computing time while processing heavy inputs such as MRI scans or to make real-time assistance possible. Particularly, filtering algorithms can greatly benefit from the parallelization the GPU architecture offers, outperforming substantially its homologous CPU implementations [16].

3 Algorithm and Implementation

As it was described in the previous section, the algorithm proposed in this work is based on the linear detector introduced in [7], with the difference that instead of processing RGB images, the proposed algorithm receives as input a single hyperspectral frame provided by a first generation Ximea MQ022HG-IM-SM5X5

snapshot camera. This camera can stream up to 170 FPS with a resolution of 2045×1085 pixels capturing 25 different spectral bands that go from 968.93 nm up to 693.74 nm. The 25 filters employed to extract each band are arranged in a mosaic pattern of 5×5 , conforming the information of a single pixel. This pattern is replicated across the entire sensor, therefore, the hyperspectral cube formed from each frame has a shape of 409×217 pixels for its spatial resolution and 25 bands for each one of them.

Once the hyperspectral cube is built from the raw frame it needs to be black/white calibrated and spectral corrected. This process can also be accelerated in a GPU platform as it is shown in the work presented by M. Villa et al. [12]. The algorithm proposed is designed to work with gray scale images so, once the calibration and the spectral correction are performed, a single band from the cube is selected and its luma is inverted so the dark contours corresponding to the vessels are marked with high brightness values. As it will be described in Subsect. 3.1, the decision on which band is to be taken is based on an optimization process to analyze which one is the most suitable for the segmentation task.

The next step is to apply the linear operators to the selected band. The operator as such, is composed by 12 different kernels, each of them defined as a zero matrix with a straight line running across the center of the matrix composed by ones, as exemplified in Fig. 1. Each one of these 12 lines is therefore intended to cover 12 possible orientations a contour could take in the image. The angular stride between each linear operator is fixed at 15° .

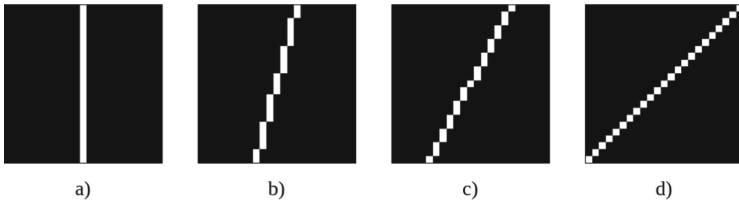


Fig. 1. Examples of different kernels at: (a) 0° , (b) 15° , (c) 30° and (d) 45°

Due to the wide range of width brain blood vessels present, one single window filter size cannot cover all its variety. To overcome this issue, two different squared window sizes are applied independently: a smaller window within the range of 11 to 15 of pixels size for detecting the capillaries and thin vessels, and a bigger window for the arteries and thick veins that covers the 15 to 31 size range. As well as the spectral band to be used, the window size of both operators is selected via optimization.

For performing the detection of any delineation that can be part of a blood vessel, every single kernel of the linear operator multiplies element-wise an aligned region of the same window size from the gray scale image. Through

this, each kernel registers the gray level that falls into its line for all the 12 possible orientations defined. Then, for evaluating which orientation is more likely to have captured an actual vessel, the average brightness of the region of the gray scale image under the position of the operator is subtracted from each average gray level captured by the kernels. In [7], the resulting value is denoted as the line strength of the kernel. The kernel with the highest strength is selected and its value is accumulated in an output image across the length and orientation of its line. This process is repeated across the entire image moving the operator as a sliding window with stride 1. Because of the summation of the strengths delivered by the operator on the output image, the sharpest contours will reach values that exceed the maximum gray level that can be represented with the bit resolution of the gray scale image. Figure 2 illustrates the process of applying one operator to the selected spectral band for obtaining the accumulated image that conforms the segmented mask.

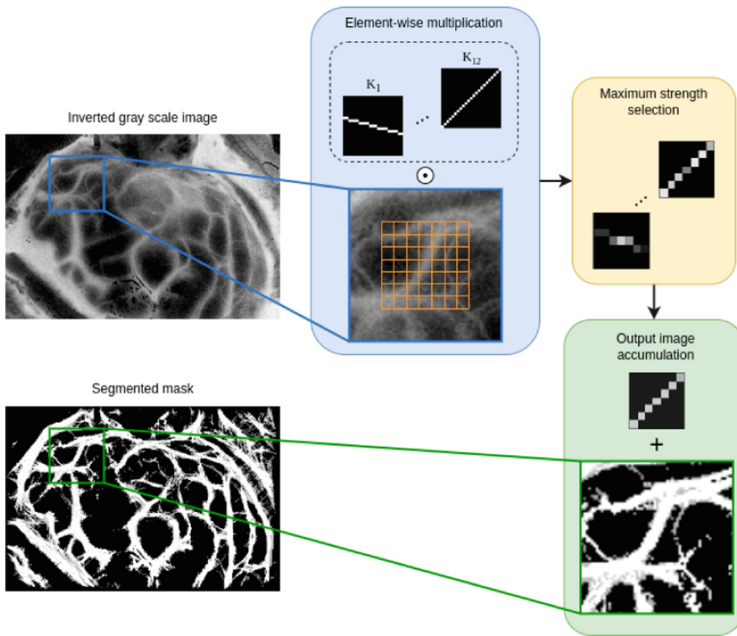


Fig. 2. Block diagram of the proposed algorithm for a single operator.

To reduce the noise introduced by the linear operator caused by partial contours, a threshold is applied to the output image by a minimum level of gray. This threshold is also fixed by the optimization process.

Since thin vessels can result in much lower strength values than thick veins or arteries, the rescaling of the output image to a range of values that can be represented with an unsigned integer, cannot be performed linearly. Given the

approximate exponential decay shape the output image histogram has, a linear rescale would cause the weaker values to be truncated to zero. Therefore, to avoid this issue, the following logarithmic correction is proposed:

$$I = \frac{\ln(S + 1) \cdot (2^n - 1)}{\ln(|\max S| + 1)} \quad (1)$$

where it is described the operations performed pixel-wise to the output image S to obtain the final image I where n is the resolution in bits.

3.1 Optimization

As described in Sect. 3, the value of certain parameters of the algorithm, such as the window size of the linear operator or the band to be processed and the thresholds of gray level for both operators, have a big impact on the output image. To ensure and accelerate the search for the best combination of parameters, their choice was made through an optimization process using the framework Optuna [17]. As it can be seen in [18] and in [19] Optuna is commonly used for optimizing hyperparameters in ML based systems for improving the training process.

The Optuna framework examines possible combinations of a set of parameters given a range to explore in a certain number of trials. For that process, it needs at least one output variable computed by the function that uses the parameters that are to be optimized so its outcome can be whether maximized or minimized. In this paper, Optuna is employed to maximize the number of vascular pixels that fall under the segmented region. Because of the particular use case addressed in this work, it is also decided to use Optuna to simultaneously minimize the tumor pixels selected. The optimization is carried out using a set of 10 hyperspectral images captured from 10 different brain tumor surgeries at Hospital Universitario 12 de Octubre in Madrid (Spain). The study was conducted according to the guidelines of the Declaration of Helsinki, and approved by the Research Ethics Committee of Hospital Universitario 12 de Octubre, Madrid, Spain (protocol code 19/158, 28 May 2019). Each hyperspectral image has its corresponding ground-truth map where not every sample but a certain amount of pixels from healthy and tumor tissue, blood vessels and dura mater are labeled. Despite the sparse content of the ground-truth, it can be calculated the average percentage of vascular and tumor ground-truth samples that have been included in the segmented area by using certain combinations of spectral band, window sizes and gray thresholds. This process is carried out through 90 trials.

Once the optimization is finished, Optuna provides the combination of parameters that delivers the highest value for the maximized metric and the set of parameters that offers the lower value for the minimized metric. Since none of these two cases are the most desired, it is necessary to analyze the trade-offs in the Pareto front that is formed with the results of the rest of the trials. Among all of the 90 calculated results, the selection is made around 3 combinations whose parameters can be seen in Table 1. Here, the 3 combinations of parameters chosen, denoted with the letters A, B and C are shown. Each combination consists

of the spectral band selected to be processed, the window size and the threshold for the gray level filtering for each operator, and the metrics obtained over the optimization set of images. In this case, the two bands selected, 15 and 16, have a wavelength of 891.79 nm and 900.40 nm respectively. As it can be seen, these 3 points are selected because they offer the best balance between high numbers of blood vessel samples segmented with low tumor error.

Table 1. Combination of parameters for the 3 selected points with their corresponding percentage of selected tissue.

Comb.	Band	Window 1	Thresh 1	Window 2	Thresh 2	Avg. Vessel	Avg. Tumor
A	16	11	266	23	274	0.81	0.08
B	15	13	120	29	348	0.85	0.11
C	16	11	266	31	274	0.90	0.15

3.2 Acceleration

One of the requirements of the algorithm is to be able to perform the segmentation at a sufficient speed that allows its integration on a real-time classification chain. Therefore, its acceleration is a central matter that conditions its usefulness. This process was carried on using a GPU programmed in the CUDA language.

The efficient parallelization of the algorithm described in 3 has certain hurdles to overcome. The usage of different kernels to each individual pixel in the image is the most resource-intensive stage of blood vessel segmentation. These kernels are considered as windows that surround each pixel. Implementing this component in a GPU is challenging due to the serial nature of this operation, which causes an overlap between the windows of various pixels. To perform this procedure in parallel, each thread is responsible for applying the filters per pixel and computing the element-wise multiplication between the filters and the window created around each pixel, followed by the computation of the strength as specified in 3. The different kernels are accessed several times during the application of the filters to the pixels of the image. The values are then stored in shared memory at the start of the process to reduce the impact of the massive memory access pattern. In this manner, the global memory access is reduced, improving memory throughput. Finally, in order to prevent race circumstances, the strength between various threads has been added in an atomic fashion, obtaining the strength accumulation per pixel at the conclusion of this process. Figure 3 depicts the differences existing between the serial implementation, where there only exists one overlapping area each iteration, and the parallel implementation, in which the calculation of these overlapping areas is performed in separate threads. Therefore, to ensure the proper accumulation, it must be necessarily atomic.

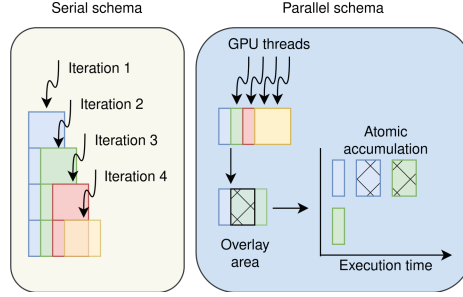


Fig. 3. Comparison between the serial implementation and the parallelization in the GPU

Thanks to there is no data-dependencies in the processing of the pixels, the other components of the segmentation process do not require any specific implementation. Instead, grid-stride loops are used in the GPU kernels in this scenario for maximum GPU performance.

4 Experiments and Results

To evaluate the linear operator under the set of selected parameters, a list of 5 hyperspectral images, different from the ones used during the optimization 3.1, were chosen. These five images also were taken during brain tumor operations at the same healthcare center than the former ones. The experiments performed are aimed 1) to prove the suitability of the linear operator as part of a classification chain that is intended to work with hyperspectral video. Therefore, 2) the accelerated algorithm must be able to achieve real-time performance not to introduce a bottleneck into the processing chain.

The experiments were conducted on two platforms: The first one, a CPU based platform consisting of a 10th generation Intel core i5-10400F working at its regular frequency of 2.9 GHz with 32 GB of DDR4 RAM. On this platform the algorithm was executed on its Python implementation with no parallelization. The second platform is a GPU (Nvidia RTX 3080) with 12GB of GDDR6X memory, 8960 CUDA cores, 384 bits of memory bus and Ampere architecture. In this case, the code executed was accelerated according to the Sect. 3.2.

4.1 Objective Results

The results presented on Table 2 show the mean percentage of vascular and tumor samples from the ground-truth detected in the segmented area and the average time in milliseconds the operator took on both platforms to generate the segmentation mask. All the results were averaged for the 5 hyperspectral images testing the 3 sets of parameters detailed in Table 1. Each one of the combinations offers a trade-off between a gain in the percentage of blood vessel

samples segmented and an increase of the tumor pixels included in the segmented mask. The line detector proposed by Ricci et al. is only implemented on CPU with a window size set to 15, as described in [7]. Band 16 is selected and the chosen gray level threshold is 266. The band and the gray threshold are taken from the set of parameters A because, as it will be seen in Subsect. 4.2, it is the combination that yields the best results. To fully evaluate the implications of the percentages shown in Table 2, its interpretation must be supported by the corresponding subjective results depicted in Fig. 5.

Figure 4 shows the synthetic RGB image extracted from a hyperspectral cube, an example of a segmented mask processed from that image and the sparse ground-truth from which the metrics of Table 2 are calculated. In the ground-truth image, black pixels refer to unlabeled samples, the green samples correspond to healthy tissue, the pink ones indicate the dura mater samples and the red and the blue pixels are used for tumor and vascular samples, respectively. The number of vein and artery samples labeled is remarkably low, especially for training any of the supervised ML algorithms mentioned in Sect. 1. Since none of them is designed to work with such sparse and scant ground truth, the inclusion of their results would make an unfair comparison with the proposed algorithm.

Table 2. Percentage of selected tissue by each combination of parameters and their computation time.

Method	Avg. Vessel	Avg. Tumor	Avg. CPU (ms)	Avg. GPU (ms)
Ricci et al.	0.47	0.02	$2.00 \cdot 10^3$	–
A	0.68	0.07	$4.94 \cdot 10^3$	5.44
B	0.72	0.18	$4.39 \cdot 10^3$	8.38
C	0.75	0.19	$4.95 \cdot 10^3$	8.83

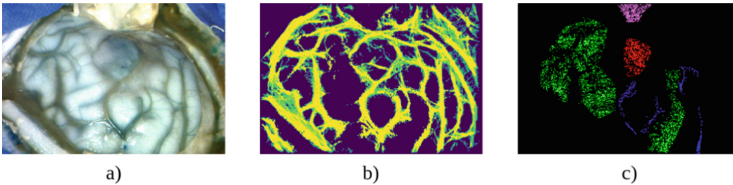


Fig. 4. (a) RGB image extracted from the hyperspectral cube, (b) Example of the strength levels in the segmentation mask for the set of parameters B, (c) Ground-truth image

The numerical results show an average speed up of 664 times of the GPU implementation over the CPU execution. Thus, achieving a performance, for the worst case, over 200 frames per second, which guarantees a real-time classification.

4.2 Subjective Results

Figure 5 shows the comparison between the original classification map and the result of overlapping on it the colorized segmented mask obtained for each one of the 3 parameters tested. Since the segmented mask is not binary, those pixels where its values are not at their maximum, are proportionally combined with the color information of the classification map.

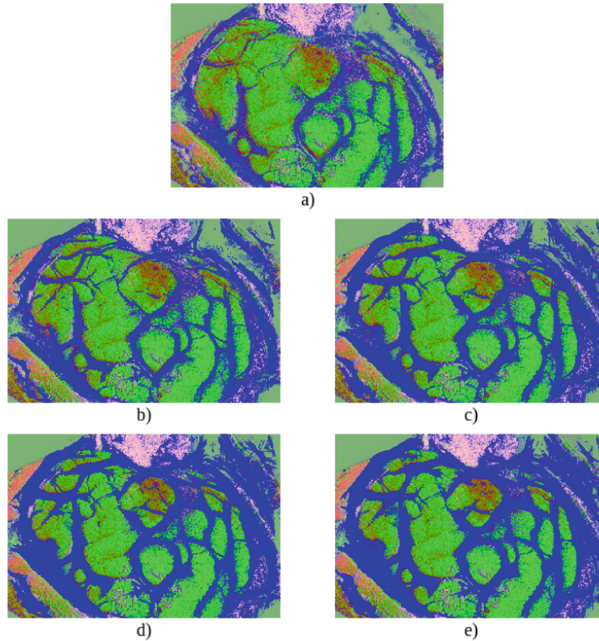


Fig. 5. (a) Original classification map, (b) masked classification map with linear detector from Ricci et al., (c) masked classification map with set of parameters A, (d) masked classification map with set of parameters B and (e) masked classification map with set of parameters C. (Color figure online)

In Fig. 5 it can be seen how the linear operator [7] improves the original classification map, but it is the method that leaves the most tumor false positives uncovered. The set of parameters A, despite having the lowest percentage of vessel samples segmented 2 among the three combinations tested, is able to mark the contours of most of the arteries and veins in a very similar way as the rest of the sets. Parameters B and C provide thicker contours around the majority of arteries and veins but do not increase the sensitivity for thinner vessels. This characteristic may imply that vessels surrounding tumor tissue are more prone to extend its limits inside the tumor area, increasing the false negative rate when detecting the tumor. Therefore, set of parameters A has proven to be the most suitable for the improvement of the classification maps, showing the robustness

acquired through the optimized selection of parameters and the use of different window sizes.

5 Conclusions and Future Work

In this work, a brain blood vessel segmentation algorithm for hyperspectral images captured during in-vivo tumor resection surgeries based on linear operators has been presented. Its capability to detect blood vessels without including samples from other kinds of tissues has been measured through objective metrics. These very metrics played a fundamental role on the parameter setting by serving as variables to be optimized. Through this optimization process it was possible to explore different candidate combinations to end up selecting the most convenient set according to the subjective observations when overlapping the segmented mask on the classification map. The combination of the algorithm with the map proved to be helpful in correcting the tumor false positives issue the classification suffers from, particularly on the contours of the blood vessels. Moreover, the proposed solution has been optimized in terms of computing performance by porting the source code to a GPU architecture. Thanks to this extent, the processing chain remains into the real-time processing constraint, i.e. 200 frames per second.

In the future, it will be explored refinement techniques for achieving smoother and better connected contours on the segmented mask. Also, current results leave room for improvement in the detection of thinner vessels and lighter contours that will be studied through filtering techniques that make use of the GPU acceleration.





References

1. Lu, G., Fei, B.: Medical hyperspectral imaging: a review. *J. Biomed. Opt.* **19**(1), 010901 (2014). <https://doi.org/10.1117/1.JBO.19.1.010901>
2. Leon, R., et al.: Hyperspectral imaging for in-vivo/ex-vivo tissue analysis of human brain cancer. In: Linte, C.A., Siewerdsen, J.H. (eds.) *Medical Imaging 2022: Image-Guided Procedures, Robotic Interventions, and Modeling*, vol. 12034, p. 1203429. International Society for Optics and Photonics, SPIE (2022). <https://doi.org/10.1117/12.2611420>
3. Urbanos, G., et al.: Supervised machine learning methods and hyperspectral imaging techniques jointly applied for brain cancer classification. *Sensors* **21**(11) (2021). www.mdpi.com/1424-8220/21/11/3827
4. Goni, M.R., Ruhaiyem, N.I.R., Mustapha, M., Achuthan, A., Che Mohd Nassir, C.M.N.: Brain vessel segmentation using deep learning-a review. *IEEE Access* **10**, 111322–111336 (2022)
5. Nazir, A., et al.: OFF-eNET: an optimally fused fully end-to-end network for automatic dense volumetric 3d intracranial blood vessels segmentation. *IEEE Trans. Image Process.* **29**, 7192–7202 (2020)
6. Babin, D., Pizurica, A., De Vylder, J., Vansteenkiste, E., Philips, W.: Brain blood vessel segmentation using line-shaped profiles. *Phys. Med. Biol.* **58**, 8041–8061 (2013)

7. Ricci, E., Perfetti, R.: Retinal blood vessel segmentation using line operators and support vector classification. *IEEE Trans. Med. Imaging* **26**(10), 1357–1365 (2007)
8. Wu, Y., et al.: Blood vessel segmentation from low-contrast and wide-field optical microscopic images of cranial window by attention-gate-based network. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*, pp. 1864–1873 (2022)
9. Fabelo, H., et al.: Deep learning-based framework for in vivo identification of glioblastoma tumor using hyperspectral images of human brain. *Sensors* **19**(4) (2019). www.mdpi.com/1424-8220/19/4/920
10. Ronneberger, O., Fischer, P., Brox, T.: U-net: convolutional networks for biomedical image segmentation (2015). arxiv.org/abs/1505.04597
11. Li, J., Udupa, J., Tong, Y., Wang, L., Torigian, D.: Segmentation evaluation with sparse ground truth data: simulating true segmentations as perfect/imperfect as those generated by humans. *Med. Image Anal.* **69**, 101980 (2021)
12. Villa, M., et al.: Data-type assessment for real-time hyperspectral classification in medical imaging. In: Desnos, K., Pertuz, S. (eds.) *Design and Architecture for Signal and Image Processing*, pp. 123–135. Springer International Publishing (2022). https://doi.org/10.1007/978-3-031-12748-9_10
13. Staal, J., Abramoff, M., Niemeijer, M., Viergever, M., van Ginneken, B.: Ridge-based vessel segmentation in color images of the retina. *IEEE Trans. Med. Imaging* **23**(4), 501–509 (2004)
14. Niemeijer, M., Staal, J., van Ginneken, B., Loog, M., Abramoff, M.D.: Comparative study of retinal vessel segmentation methods on a new publicly available database. In: *SPIE Medical Imaging* (2004)
15. Eklund, A., Dufort, P., Forsberg, D., LaConte, S.M.: Medical image processing on the GPU - Past, present and future. *Med. Image Anal.* **17**(8), 1073–1094 (2013). www.sciencedirect.com/science/article/pii/S1361841513000820
16. Wang, X., Shi, B.E.: Gpu implementation of fast Gabor filters. In: *Proceedings of 2010 IEEE International Symposium on Circuits and Systems*, pp. 373–376 (2010)
17. Akiba, T., Sano, S., Yanase, T., Ohta, T., Koyama, M.: Optuna: a next-generation hyperparameter optimization framework (2019). arxiv.org/abs/1907.10902
18. Martin-Perez, A.: Hyperparameter optimization for brain tumor classification with hyperspectral images. In: *5th Euromicro Conference on Digital System Design (DSD)* (2022)
19. Srinivas, P., Katarya, R.: hyoptxg: optuna hyper-parameter optimization framework for predicting cardiovascular disease using xgboost. *Biomed. Signal Process. Control* **73**, 103456 (2022). www.sciencedirect.com/science/article/pii/S1746809421010533



Neural Network Predictor for Fast Channel Change on DVB Set-Top-Boxes

Tomás Malcata^{1,2,3} , Nuno Sebastião^{4,5} , Tiago Dias^{1,4,5} ,
and Nuno Roma^{1,2,3} 

¹ INESC-ID, Lisbon, Portugal

`tomas.malcata@tecnico.ulisboa.pt`, `nuno.roma@inesc-id.pt`

² Instituto Superior Técnico, Lisbon, Portugal

³ Universidade de Lisboa, Lisbon, Portugal

⁴ Instituto Superior de Engenharia de Lisboa, Lisbon, Portugal

`{tiago.dias,nuno.sebastiao}@isel.pt`

⁵ Instituto Politécnico de Lisboa, Lisbon, Portugal

Abstract. With the generalization of digital Television (TV), keeping the channel change delay as low as possible gradually became a difficult requisite in what concerns the resulting user's Quality-of-Experience (QoE). Frequently, this latency may be higher than 2s. While many state-of-the-art Set-top-Boxes (STBs) already include a shadow tuner to anticipate the tuning of the next channel, they strive to predict which channel should be pre-tuned, generally opting for one of the adjacent channels. The presented research proposes the use of a predictive system to assist the STB in the forecast of the channel(s) the user will select next. The implemented predictor is based on a Recurrent Neural Network (RNN) and makes use of STB log data concerning the user's channel changes history to train (and adjust) the model every week. To attain this objective, the most convenient hyperparameter combination that not only fulfilled the aimed prediction accuracy but also suited the rather limited computational constraints of most current STBs had to be identified. The obtained experimental results, validated using four embedded processor families commonly equipping commercial STBs, showed a prediction accuracy of 50.2% for a single-channel prediction and 67.7% when five channels were simultaneously predicted. When combined with the existing dual-tuning system of current STBs, the proposed predictor can save as much as 1000s per month in TV channel change delays, greatly improving the resulting user's QoE.

Keywords: Channel Change Delay · Set-Top-Box · Predictive System · Recurrent Neural Network

1 Introduction

Digital Video Broadcasting (DVB) systems, either DVB-C, DVB-T or DVB-IPTV, typically suffer from a high channel change (zapping) time. This results from the inherent limitations of transmitting the data at a constant bitrate, on a broadcast method (in which there is no ability to request additional data) and having only specific points in time where decoding can start (due to the video compression techniques). Such limitations make the total zapping time range from a couple of seconds to tenths of seconds, which significantly reduces the user's Quality-of-Experience (QoE) when zapping through the channels.

Operators that control their delivery networks, typically those that have DVB-C and DVB-IPTV networks, commonly rely on custom built specialized hardware devices installed in the customers premises, denoted as Set-top-Boxes (STBs), to deliver their services. Operators may decide to include in these devices additional features that allow to reduce the channel change time. Nevertheless, due to cost reasons, such embedded devices are commonly restricted both in terms of these additional features as well as in their computational capabilities.

Various methods have been devised to reduce the channel change time [3, 5, 8]. However, all of them have an additional cost to the operator, either because it requires additional hardware resources or additional bandwidth in the network. Since these resources are limited, the best QoE is achieved when the user actually changes to the channel(s) for which the system was previously optimized.

Using prediction systems based on Neural Networks (NNs) [6] has now become more feasible, which allows to more accurately predict the channels that the user will watch next. With this additional information it becomes possible to dynamically configure the STB in order to achieve the previous goal (i.e., provide the smallest possible delay to change to the desired channel), by configuring the local system resources to preemptively get the data needed to start playback of a new channel. Furthermore, it is preferable to have the prediction system using only the already available computational resources of the STB, as opposed to having a centralized solution, due to the smaller cost. However, this means that the computational load of the predictor should be light enough so that it is compatible with the limited computational resources that exist in the STB.

This paper presents a method that predicts the channel change behavior of a given user based on a Recurrent Neural Network (RNN) that suggests a list of channels that the user will most probably watch at any given moment. Such RNN can be trained and executed entirely in the computationally restricted environment of a STB. In Sect. 2, an overview of the related work is provided while Sect. 3 presents the proposed solution to predict the user's channel change behavior. Section 4 presents the assessment of the proposed model both in terms of its prediction accuracy and of the computational requirements on various typical embedded platforms, similar to those used in STBs. Finally, some conclusions are drawn in Sect. 5.

2 Related Work

The channel change time is the result of the sum of various delays that can be divided into: i) distribution network access delay, i.e., the carrier frequency tuning time, for DVB-C or DVB-T, or the typical multicast group join time of DVB-IPTV; ii) synchronization delay, i.e., the time it takes until a random access point in the transport stream is received; iii) video buffering delay, i.e., the time it takes to fill the buffers with the data broadcasted at a constant bitrate to a level at which playback can start; and iv) device processing delays, i.e., key press event handling and internal processing - typically quite small when compared to the other delays. The sum of these delays may vary from a couple of seconds to tenths of seconds, with the synchronization and buffering delays being the largest contributors.

Some STBs possess dedicated hardware to assist in reducing the channel change time, e.g., by having the ability to simultaneously capture multiple streams, demultiplex various channels and even the ability to decode more than one video stream. The channel change time when using all of these capabilities, including simultaneous video decoding, allows the channel change time to be as low as 20 ms. However, since these capabilities come at a cost, the actual number of channels that can be simultaneously processed is quite limited.

Besides the use of additional dedicated hardware, it is also possible to reduce the channel change time by having additional companion streams or an additional medium to fetch the required data to fill in the video buffers in a faster way. This additional medium is most commonly a network connection, which is nowadays a common feature even on the STBs of the DVB-C operators.

Regardless of the technique used to reduce the channel change time, the amount of resources is always limited and thus requires the system to select which channel (or channels, depending on the available resources) it should prepare to change to. Ideally, the system would always be prepared to change to the channel that the user will watch next. Furthermore, when looking at solutions that require additional bandwidth to support a faster channel change (e.g., additional dedicated streams or a network connection), it is important to note that these also impose an increased cost to the operator and should be minimized.

Furthermore, whether the distribution network is DVB-C, DVB-T or DVB-IPTV, the use of prediction systems to determine the next watched channel significantly helps in making a better use of the available resources to reduce the channel change time [1, 5, 8, 12], either by configuring the local system resources (e.g., demuxer or video decoder) [4] or by preemptively getting the data from the network to provide the streams with optimal delay [3].

3 Proposed Solution

3.1 RNN Model

NNs [6, 9] present several advantages also for the development of channel change prediction systems. Firstly, the inference procedure in NNs is very fast, which

is quite important to minimize the delay when choosing the next channel to be displayed in a channel change event. Secondly, NNs provide high tolerance to failure even if some input data is incomplete. This feature is quite relevant for STB-oriented prediction systems, since the input data (i.e., the users' channel changes history) is stored in logs that might not always contain all the required data due to unexpected events, such as log corruption or lack of disk space. The capability of a NN to adapt its behavior dynamically is another important aspect, since it is a crucial feature when dealing with data collected in real-time. For the channel change prediction problem, it is well known that users quite often change not only their favorite channels list but also their viewing schedules. A NN model can easily adapt to such changes simply by retraining the network, i.e., adjusting the neurons weights. This makes it possible to retrain the model periodically with some new data (e.g., weekly), without having to discard the existing model and recreate a new one from scratch. Finally, RNNs allow having the model output depending on sequential data, which is quite advantageous to significantly improve the accuracy of the prediction in time series problems [9], like the channel switching sequences of a user.

For the aforementioned reasons, the channel change prediction method herein proposed is based on a RNN model with a many-to-one configuration, where the channel prediction model takes into consideration not only the current date, time and Television (TV) channel the user is watching for channel change events but also a set of channels previously displayed to significantly improve the accuracy of the prediction. Although other existing models consider additional user data [2, 5, 8, 10], such as the user preferred TV genres, favourite programs, surfing behaviour, demographic information, etc., we choose to adopt a simpler model tailored for implementations on STBs, due to the tight computational performance, Random Access Memory (RAM) and persistent storage constraints of these platforms.

The architecture of the proposed RNN model is shown in Fig. 1, where 'Day of the year', 'Time of the Day' and 'Week Day i ' correspond to the instant the channel change event occurred, 'Previous channel j ' are the most recent j channels watched by the user at such instant, and 'Output Channel k ' is an ordered list of k channels that the user is most likely to switch into, arranged from the highest to the lowest probability. To reduce the propagation of errors, the 'Day of the year' and 'Time of the Day' values are normalized to year days and seconds, respectively. Conversely, the 'Week Day i ', 'Previous channel j ', and 'Output Channel k ' values are treated as classes and one-hot encoded. Consequently, the model has $9+C$ input nodes and C output nodes, where C is the number of recent channels watched by the user that are considered for the prediction. The number of nodes in each hidden layer is also C , as shown in Fig. 1. The proposed RNN model considers a maximum of 50 channels ($C = 50$), since recent studies show that most users do not watch more than 50 different channels [12].

The amount of hidden layers, the number of nodes in each hidden layer, the type of neurons, and the unroll length (i.e., the length of the input sequence containing the history of the last channels watched by the user) are the model

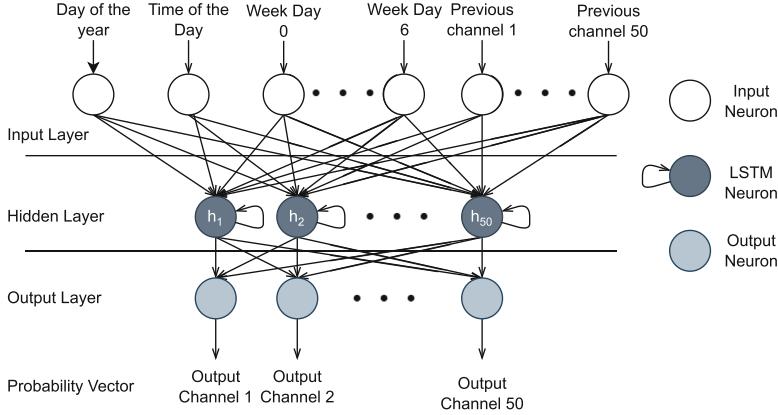


Fig. 1. Architecture of the proposed RNN.

hyperparameters influencing the RNN architecture, as discussed in Sect. 4.2. The remaining model hyperparameters are the learning rate, the dropout rate, the number of weeks and epochs used to train the network.

3.2 Model Implementation

The proposed RNN model was implemented using the KANN framework [7], which is an open-source standalone and lightweight deep learning library developed using the C programming language. When compared to other state-of-the-art frameworks, KANN is as efficient as most of them for small NN, like the RNN model herein proposed. Nonetheless, KANN presents important advantages for implementations on constrained embedded platforms, such as STBs. In particular, the computational efficiency of this framework is very high, due to being implemented using a low-level programming language. For the same reason, its RAM and persistent storage requirements are quite modest. For example, the framework contains only four files that occupy 132 KB in the filesystem of an embedded platform. Furthermore, KANN has no software dependencies (except for the C standard library), which makes it highly portable and suitable for implementations in most Unix-based computational systems, such as STBs.

Using the KANN framework, the proposed RNN model was implemented with a specially developed C function based on the following API functions: `kann_layer_inputs()`, to receive the model inputs; `kann_layer_rnn()`, `kann_layer_lstm()`, and `kann_layer_gru()`, to implement the hidden layer(s); `kann_layer_dropout()`, to perform the dropout regularization; and `kann_layer_cost()`, to select the output(s). The inputs to such functions are the RNN model hyperparameters, i.e., the number of inputs, outputs, and hidden layers, the amount of neurons per hidden layer, the neurons' types, and the dropout rate.

To train the proposed RNN model another custom C function was developed, since the KANN API only provides a training function for simple Feedforward

Neural Networks (FNNs). Such training procedure encompasses two parts: *setup* and *action*. In the *setup* part, all the information necessary to conduct the training procedure is obtained, i.e., the amount of inputs, outputs, and trainable variables of the model. Then, a new NN model is created by unrolling the initial RNN model through time and turning it into an equivalent FNN. The training is conducted using this FNN model and the Backpropagation Through Time (BPTT) algorithm [11] and the resulting parameter values are used to configure the original RNN model. Finally, in the *action* part, the model weights are updated as a result of the training procedure.

4 Experimental Evaluation

4.1 Considered Datasets

The datasets used in this work, from which the inputs to the model were obtained, consist of channel change events, considering the day of the week and the time at which those events occurred. Such events are directly obtainable in the STBs from the users' channel change actions (e.g., channel up button press).

For the analysis presented in this paper, two main datasets were considered: i) a synthetic dataset, mainly used to make an initial assessment of the model and its performance; and ii) a real user dataset, which represents the actual behaviour of a set of STB users. From the real user dataset, a subset of 30 users was also used to determine the hyperparameters of the model.

The synthetic dataset was synthesized to represent an artificial user that has a mostly consistent and predictable zapping pattern (i.e., a user that repeatedly watches the same channel on various days at the same time - e.g., to watch the prime time news program that airs every night at 8 p.m.) that repeats every week. To achieve this goal, two types of channel change events were added to this dataset: i) recurrent channel change events and ii) random channel change events. The recurrent channel change events are those that occur when the user is starting to watch his routine events (e.g., the news). These occur at similar hours and happen every week (with slight variations on the actual channel change time, e.g., +/-30 minutes). The random channel events represent the other channels that the user watches. For this dataset, the channel changes for a period of two months were synthesized, which made a total of 227 events. Due to the random channel events, the prediction accuracy when using this synthesized dataset is not 100% but is still quite high (85%), as shown in Table 2.

The real user dataset is comprised of the actual channel change events of 300 distinct STB users throughout a two months period. This dataset was split into two subsets: one that is used to determine the best hyperparameter values for the RNN, which is composed of the data pertaining to 30 users, and a second that is used to assess the performance of the proposed solution, which is composed of the data pertaining the remaining 270 users.

The 30 users subset, which is used to determine the hyperparameters of the RNN, is comprised of specifically selected users whose behaviour causes the RNN to have a higher accuracy (denoted as best 30 users dataset). The selection of the

users to be included in this dataset is based on a random and iterative process. This process starts by creating 100 distinct RNN models by randomly varying the hyperparameters. Then, each user from the real users dataset is randomly assigned to one RNN model and its accuracy is determined. Afterwards, the 5 users with the lowest accuracy are discarded. This process is repeated until 30 users are left, which will comprise the best 30 users dataset.

The approach to use only a subset of the users to determine the hyperparameters was preferred since not only it reduces the effort to find a good combination of hyperparameters but also contributes to exclude users that have no clear pattern from affecting the RNN model. Moreover, given the available dataset, it allows testing how a model generalises to other users.

4.2 Network Parameterization

Based on the defined model and using the best 30 users dataset, the RNN hyperparameters that are best suited to generate models to predict the next channels were determined. For each hyperparameter, various RNNs were generated (one for each possible value of a given hyperparameter). Each of these RNNs were then used to predict the next channels for each user in the best 30 users dataset and its accuracy was determined. It is relevant to note that a given hyperparameter may result in a higher accuracy for only a specific user. Hence, besides the average accuracy of the 30 users for a given hyperparameter value, the number of users that present a better accuracy for that hyperparameter value was also taken into account when deciding which hyperparameter to adopt. As part of the criteria to choose the best value for each of the hyperparameters, the training time and the model complexity (measured as the number of trainable parameters) were also considered, due to the fact that these are of particular relevance when considering that the training phase is executed in a device with limited computational capabilities, as is the case of a STB.

The evaluated hyperparameters were the following: dropout rate, minimum Root Mean Square Error (RMSE), network type, number of hidden layers, unroll length, and number of weeks used in the training. The dropout rate is responsible for freezing some neurons (a percentage equivalent to the dropout rate) during the training phase, which counteracts the over-fitting of the RNN. The minimum RMSE is used as one of the two stop conditions of the chosen RNN (the other being a maximum of 1500 epochs per train); it represents the risk level of the RNN - if it is low, then the RNN may perform worst with the test dataset. The network type (identified by its neuron type) has a significant effect on the training time and the number of overall parameters, with the Long Short Term Memory (LSTM) being the most complex cell and the vanilla RNN being the least complex cell. The number of hidden layers used in the model linearly increases the number of parameters that require training. A higher number of hidden layers has the consequence of increasing the training time as well as the memory requirements of such model. Hence, it is advantageous to keep this value to the minimum possible. The unroll length determines the length of the sequence of inputs that will influence the output. In this case, it defines the number of channel changes that will influence the next channel to be watched

Table 1. Hyperparameters - assessed values and chosen values (depicted in bold and underline): # parameters - the amount of parameters to train in the model ($\times 10^3$); Train time - the time it takes to train 100 samples (in seconds); Accuracy - the average accuracy of the 30 users (in %); # users - the amount of users of the 30 user dataset that obtain better results with the given hyperparameter value

	Dropout rate value				Minimum RMSE					Neuron type			Weeks used to train				
	<u>0.5</u>	0.6	0.7	0.8	0.1	0.2	0.3	0.4	<u>0.5</u>	<u>LSTM</u>	GRU	Vanilla RNN	1	2	3	4	<u>5</u>
# parameters	14.1	14.1	14.1	14.1	14.1	14.1	14.1	14.1	14.1	14.1	10.9	4.5	14.1	14.1	14.1	14.1	14.1
Train time (s)	145	142	148	149	278	272	245	234	235	159	204	82	17	62	106	131	153
Accuracy (%)	60.2	59.4	58.7	54.1	61.8	61.5	61.8	62.1	61.9	60.3	56.2	53.6	44.4	51.5	55.6	59.0	60.4
# users	14	4	9	1	4	7	4	7	8	22	6	2	0	0	0	7	23
	Unroll length							Learning rate					Number of layers				
	2	3	<u>4</u>	5	6	7	8	<u>0.01</u>	0.03	0.05	0.07	0.09	0.11	<u>1</u>	2	3	4
# parameters	14.1	14.1	14.1	14.1	14.1	14.1	14.1	14.1	14.1	14.1	10.9	4.5	14.1	14.1	26.5	3.9	51.2
Train time (s)	161	147	152	204	255	304	373	159	212	228	253	273	314	159	623	935	963
Accuracy (%)	59.6	60.6	60.1	59.7	59.7	59.4	58.0	60.3	60.4	60.8	60.0	59.2	59.1	60.3	55.6	48.6	44.1
# users	6	7	6	4	3	3	1	6	5	8	2	3	6	24	5	1	0

by a given user. The learning rate value affects how fast the model can change and, consequently, it influences how fast the model converges and whether or not it reaches the minimum RMSE risk or not. For this hyperparameter, it was preferred to reduce the training so that it can be better fitted to the restrictions of the embedded system for which it is targeted. The number of weeks used for training does not influence the RNN model, its training mechanism or its hyperparameters. However, it has a critical role in defining the training dataset, since it is not viable to store all channel change history in the STB from its initial set up and use it all to train the RNN.

To determine the best value for each of the RNN hyperparameters, a set with an initial random value for each hyperparameter was created. From this base set, other sets with different values for the hyperparameter under assessment were created. Subsequently, for each of those sets, the corresponding model was created and evaluated for each user in the best 30 user dataset. The hyperparameter value that provided the best performance was selected and the process continued with the next hyperparameter under assessment.

Table 1 shows a summary of the various hyperparameters, the used values in the assessment phase and the ones chosen as the model's hyperparameter values.

4.3 Accuracy Results

After having settled the RNN hyperparameters, the proposed solution was assessed using both the synthetic dataset - used to determine if the model behaves as expected - and the real user dataset. For the later analysis, the 270 real users dataset was used as well as the complete 300 users dataset. For each user, the accuracy is calculated on a per-week basis. Initially, only the first week of data is considered in the training and for the second week, the accuracy of the prediction is obtained. Subsequently, in the third week, the data from the

Table 2. Accuracy and memory usage results for the model’s execution on the used datasets: Artif - the artificial user dataset; Real users - the dataset which contains the events for 300 real users; 270 u - the subset containing 270 users of the real users dataset; 30 u - the subset of the best 30 real users used in the hyperparameter parameterization; all - all of the real users dataset (300); best - the user with the highest accuracy in the real users dataset; worst - the user with the lowest accuracy in the real users dataset

#Channels	Accuracy						Max mem (MB)	
	Artif	Real users					Artif	Real users
		270	30 best	All	Best	Worst		
1	83.5	49.3	57.3	50.2	81.6	10.7	0.5	2.4
2	88.3	55.3	69.0	56.9	83.7	17.0		
3	93.1	59.8	75.5	61.4	86.8	21.9		
4	94.9	63.0	80.2	64.8	88.3	25.5		
5	95.2	65.7	83.8	67.7	90.2	29.0		

previous two weeks is used to train and the accuracy is obtained for the events in the third week. This process repeats itself until the full two months of data is considered. At the end, the average accuracy for each user is gathered and an average of the accuracy of all the users is presented.

Besides the average accuracy of the artificial user and of the real users, the accuracies for the worst user and the best user are also presented. In this situation, the worst user is the one for which the proposed solution does not improve much the channel change time, whereas the best user shows the model’s performance with the highest prediction accuracy.

Table 2 shows the accuracy for the various datasets as well as the maximum amount of used RAM. It is possible to observe that the proposed model achieves a significantly high prediction success for the 270 real users dataset. It is also possible to observe that for the 300 real user dataset, the accuracy is slightly higher, however this dataset includes the users that were initially chosen to determine the model’s hyperparameters, which inherently have a higher accuracy.

As expected, the artificial user dataset presents the highest accuracy values for the various channel configurations due to the way that this user was synthesized to have a regular and predictable behaviour. For the real users, it is worth noting that the model was able to have an average accuracy of 49.3% when considering the prediction of just one channel for the 270 users dataset (the most significant user group) and of 65.7% when considering a prediction of five channels. When considering the whole 300 real user dataset, the average accuracy is 50.2% for one channel (ranging from 81.6% to 10.7%) up to 67.7% for five channels (ranging from 90.2% to 29.0%).

4.4 Execution Performance and Required Resources

To validate and evaluate the viability of the proposed prediction mechanism, the developed model was executed in four different off-the-shelf embedded platforms,

Table 3. Embedded platforms considered in the conducted performance evaluation.

	ARM1176JZF-S	Cortex-A9	Cortex-A53	Cortex-A72
Frequency	700 MHz	866 MHz	1.2 GHz	1.5 GHz
# Cores	1	2	4	4
DRAM	512 MB	497 MB	1 GB	8 GB

Table 4. Average training times (in seconds) in the considered embedded platforms.

	ARM1176JZF-S	Cortex-A9	Cortex-A53	Cortex-A72
Artificial user	0.91	0.30	0.19	0.20
Best user	303	133	72	14
Best 30 users	1614	546	506	310
Worst user	4608	4501	4281	1705
All users	3888	2758	1465	627

equipped with ARM processors commonly adopted by commercial STBs. Table 3 presents the characteristics of each considered embedded platform.

The executable binary and required libraries occupy 401 KBytes, which can be easily accommodated in the filesystem of any of these platforms. Since the training data is obtained directly from the existing log files and related data structures of the STB, it does not require any relevant added storage space. In what concerns the system memory (DRAM), it was observed that all conducted experiments did not require more than 2.4 MB. This value represents the amount of memory needed to allocate the RNN and the training dataset. As it can be also observed in Table 3, this (peak) memory requirement is easily satisfied by the considered embedded platforms and for most current STBs. It should be noted that this value is independent of the number of channels being predicted because the model runs only once and returns an ordered list of channels from which it is chosen the number of channels to consider.

Table 4 presents the observed training times for each of the considered embedded platforms. As it would be expected, the obtained performance is highly dependent on the processor family and on the corresponding operating frequency.

To ensure the best user’s QoE, the considered setup assumed that the training procedure is executed weekly (thus conforming with the contents periodicity that is usually adopted by most TV broadcasting networks) and it was scheduled to a period of the day when the STB is less likely to be used for TV playback (i.e., 5 a.m.). Naturally, such scheduling can be easily tuned to each particular user profile based on the observation of its routine. Furthermore, and to avoid any possible perturbation to the user’s QoE, a maximum timeframe was considered for the whole training procedure (i.e., 90 min). As it can be observed, all the considered models (apart from the rather unrealistic worst user scenario) do not saturate at this stop condition - even when executing in the most restricted

Table 5. Comparison of the proposed model with other models referred in the literature in what concerns the obtained average accuracy [%].

#Channels	Up&Down[3]	J48 Default[1]	Ada Boost[1]	CL[12]	SL[12]	Proposed
1	12.2	37.4	37.4	19.3	22.2	50.2
2	23.5	–	–	–	–	56.9
3	31.2	–	–	23.8	48.0	61.4
4	–	–	–	–	–	64.8
5	–	–	–	54.8	63.6	67.7

computing platform, presenting an execution time that rarely exceeds an entire hour (3600s). In particular, it was observed that 10 to 30 min is more than enough to run this training procedure for the most regular users. Therefore, the training parameters and the resulting accuracy validate the feasibility of this model in these embedded platforms and opens space to define other system-optimized criteria to balance between the desired accuracy, the resulting saved time upon a channel change, and the cost of fetching the required number of channels.

4.5 Comparison with Other Approaches

Table 5 compares the proposed model with other state-of-the-art models. For the models presented in [1] and [12], the accuracy was calculated using the datasets considered in the respective papers. Nevertheless, such results are expected to be very similar to those that would be obtained if the dataset used in the proposed model was used. For the Up&Down model [3], the presented results are a summary of the results reported by the authors but only considering the best combination of one, two, or three channels to be predicted. This last accuracy was obtained using the same dataset as the one used to train the proposed model.

As it can be observed, the proposed RNN model, specifically implemented for execution environments with low computational resources, clearly outperformed both the Up&Down model and the other tree-based models. Also, the accuracy of the proposed model is significantly higher than the one presented in [12] for one single channel prediction, although it is very similar to a five channels prediction. This is primarily due to the similarity of these two models, since both are LSTM based. Furthermore, it also shows that the proposed low-resource implementation for embedded environments based on the Kann framework [7] does not impact the resulting accuracy, since slight improvements could even be achieved when compared with the model trained without performance restrictions.

5 Conclusions

A new predictive system based on a RNN model was proposed to assist the tuning mechanism of current STBs in the forecast of the channel(s) a user will select

next, in order to reduce the involved channel change delay. The implemented predictor makes use of log data concerning the user's channel changes history to train (and adjust) the model. Considering the tight computational constraints of most current STBs, the most convenient hyperparameter combination that still fulfills the aimed prediction accuracy had to be identified. The obtained results, validated using four embedded processor families commonly equipping commercial STBs, showed a prediction accuracy of 50.2% for a single-channel prediction and 67.7% when five channels were simultaneously predicted. When combined with the existing dual-tuning system of current STBs, the proposed predictor can save as much as 1000 s per month in what concerns the TV channel change delay, greatly improving the resulting user's QoE.

Acknowledgements. This work was partially supported by national funds through Fundação para a Ciência e a Tecnologia (FCT) under project UIDB/50021/2020.

References




1. Basicovic, I., Kukolj, D., Ocovaj, S., Cmiljanovic, G., Fimic, N.: A fast channel change technique based on channel prediction. *IEEE Trans. Consum. Electron.* **64**(4) (2018). <https://doi.org/10.1109/TCE.2018.2875271>
2. Cha, M., Rodriguez, P., Crowcroft, J., Moon, S., Amatriain, X.: Watching television over an IP network. In: Proceedings of the ACM SIGCOMM Internet Measurement Conference, IMC (2008). <https://doi.org/10.1145/1452520.1452529>
3. Cho, C., Han, I., Jun, Y., Lee, H.: Improvement of channel zapping time in IPTV services using the adjacent groups join-leave method. In: 6th International Conference on Advanced Communication Technology: Broadband Convergence Network Infrastructure, vol. 2 (2004). <https://doi.org/10.1109/icact.2004.1293012>
4. Fimic, N., Basicovic, I., Teslic, N.: Reducing channel change time by system architecture changes in DVB-S/C/T set top boxes. *IEEE Trans. Consum. Electron.* **65**(3) (2019). <https://doi.org/10.1109/TCE.2019.2913361>
5. Kim, Y., et al.: Reducing IPTV channel zapping time based on viewer's surfing behavior and preference. In: IEEE International Symposium on Broadband Multimedia Systems and Broadcasting 2008, Broadband Multimedia Symposium 2008, BMSB (2008). <https://doi.org/10.1109/ISBMSB.2008.4536621>
6. LeCun, Y., Bengio, Y., Hinton, G.: Deep learning. *Nature* **521**, 436–44 (2015). <https://doi.org/10.1038/nature14539>
7. Li, H.: kann: a lightweight C library for artificial neural networks. <https://github.com/attractivechaos/kann>
8. Ramos, F.M., Crowcroft, J., Gibbens, R.J., Rodriguez, P., White, I.H.: Reducing channel change delay in IPTV by predictive pre-joining of TV channels. *Signal Process. Image Commun.* **26**(7) (2011). <https://doi.org/10.1016/j.image.2011.03.005>
9. Sherstinsky, A.: Fundamentals of recurrent neural network (RNN) and long short-term memory (LSTM) network. *Physica D* **404**, 132306 (2020). <https://doi.org/10.1016/j.physd.2019.132306>

10. Tongqing, Q., Zihui, G., Seungjoon, L., Jia, W., Qi, Z., Jun, X.: Modeling channel popularity dynamics in a large IPTV system. In: SIGMETRICS/Performance'09 - Proceedings of the 11th International Joint Conference on Measurement and Modeling of Computer Systems, pp. 275–286 (2009). <https://doi.org/10.1145/1555349.1555381>
11. Werbos, P.J.: Backpropagation through time: what it does and how to do it. *Proc. IEEE* **78**(10), 1550–1560 (1990). <https://doi.org/10.1109/5.58337>
12. Yang, C., Ren, S., Liu, Y., Cao, H., Yuan, Q., Han, G.: Personalized channel recommendation deep learning from a switch sequence. *IEEE Access* **6**, 50824–50838 (2018). <https://doi.org/10.1109/ACCESS.2018.2869470>

Hardware Architectures and Implementations



AINoC: New Interconnect for Future Deep Neural Network Accelerators

Hana Krichene^(✉), Rohit Prasad, and Ayoub Mouhagir

Université Paris-Saclay, CEA, List, 91120 Palaiseau, France
{hana.krichene, rohit.prasad, ayoub.mouhagir}@cea.fr

Abstract. Data exchanges can be significant in the Deep Neural Network (DNN) algorithms. The interconnection between computing resources can quickly have a substantial impact on the overall performance of the architecture and its energy efficiency. Similarly, access to the different memories of the system, with potentially high data sharing, is a critical point. To overcome these problems, in this paper, we propose a new interconnect network, called AINoC, for future DNN accelerators, which require more flexibility and less power consumption to facilitate their integration into artificial intelligence (AI) edge systems. AINoC is based on (1) parallel routing that ensures communication/computation overlap to improve performance and (2) data reuse (filters, image inputs, and partial sums) to reduce multiple memory accesses. In the experiment section, AINoC can speedup LeNet5 convolution layers by up to $71.74\times$ in latency performance w.r.t. a RISC-V-based CPU and also speedup MobileNetV2 convolution layers by up to $2.35\times$ in latency performance w.r.t. a dataflow architecture featuring row-stationary execution style. AINoC provides any-to-any data exchange with wide interfaces (up to 51.2 GB/s) to support long bursts (up to 384 flits/cycle with packed data, i.e., $3 * 8$ -bit data in a 32-bit wide datapath) while executing LeNet5 and MobileNetV2. AINoC supports flexible communication with many multicast/broadcast requests with non-blocking transfers. Parallel communication in AINoC can provide up to $128\times$ more throughput (flits/cycle) and bandwidth (GB/s), using parallel routing with respect to single-path routing while executing convolution layers of LeNet5 and MobileNetV2.

Keywords: On-Chip Interconnect · Network-on-Chip · DNN · CNN · dataflow execution · parallel processing

1 Introduction

Deep Neural Network (DNN) algorithms and, in particular, Convolutional Neural Network (CNN) algorithms [1] have become good candidates for Machine Learning (ML) and Artificial Intelligence (AI) research. Recent CNNs are composed of multiple layers with billions of parameters [2] and complex computational algorithms. This requires consequent memory and computation resources to be performed efficiently on existing hardware architectures that are limited

in terms of resources & energy, and also suffer from low flexibility & scalability. These hardware accelerators are mainly based on many Processing Elements (PEs) involving optimised operators for Multiplication-ACcumulation (MAC) execution and local buffers for storing data, which are frequently reused as filter parameters or intermediate data [3].

When designing a CNN accelerator, communications between the PEs themselves, and between the PEs and the memory are essential to consider. Therefore, the on-chip communication device must be carefully designed to exploit many PEs and the peculiarities of CNN algorithms, which allow for both performance and energy efficiency improvements. Furthermore, this communication device must provide robust any-to-any data exchange with large interfaces to support long data bursts. It must also support parallel, flexible and energy-efficient communications to accommodate multiple simultaneous communication requests (multicast/broadcast) without blocking transfer(s).

To achieve an efficient and scalable algorithm-architecture matching, we propose in this paper a novel interconnect network AINoC for the future CNN accelerators. This network consists of a set of routers optimised for parallel dataflow processing with minimal data transfer cost to achieve energy-efficient CNN processing without compromising accuracy and application performance. AINoC is designed to (i) provide parallel communications of different types of shared data (weights, Input feature maps and Partial sums) in CNNs, (ii) reduce the cost of data exchange so as not to degrade performance, and finally (iii) facilitate the processing of different CNNs and different layers of the same network (Convolution, Fully Connected, Point-Wise, Depth-Wise, Residual, etc.).

The remainder of this article is as follows. Section 2 describes the state-of-the-art interconnect networks in CNN accelerators. Section 3 presents the architecture of the AINoC network and its features. Section 4 introduces the evaluation environment and presents the experimental results. Finally, Sect. 5 concludes the paper.

2 Related Work

Using an interconnect network in dataflow CNN architectures becomes essential to efficiently manage data transfers in the PE grid and memory accesses. These interconnect networks must ensure parallelism, data reuse and flexibility/scalability. Several works have proposed different communication solutions in many neural network accelerators. The work [3] proposes a Row Stationary dataflow model that minimises energy consumption and data movement. The Eyeriss architecture allows fully exploiting spatial parallelism and data reuse in CNNs. This is achieved through two communication networks. The first one is based on a bus to ensure multicasting of weights and input feature maps (ifmaps). The second one is based on neighbourhood connections to facilitate the accumulation of partial sums. However, the solution proposed in this work cannot support all CNN layers (only the CONV layers). This non-flexibility of the architecture makes it challenging to implement algorithmic innovations and

strongly reduces the scalability of the architecture. The DianNao family [4], with its different versions ShiDianNao [5] and DaDianNao [6], proposes neural architectures composed of Neural Functional Units (NFUs), which process the CNN layers by organising the dataflow through the NFUs. DaDianNao uses two types of interconnect networks: a mesh network and an H-Tree network to connect its neural processing units. Besides, DaDianNao implements a memory on-chip called eDRAM (or embedded DRAM) to reduce memory access latency. ShiDianNao exploits the principle of data reuse by using the Output Stationary dataflow model that reduces the energy consumption necessary to read/write partial sums by accumulating these sums locally at the PEs level. Despite the diversity of communication devices in these architectures, data transfers in DianNao remain expensive and inefficient, DaDianNao ignores data reuse, and ShiDianNao is optimised for small CNNs, which reduces its configurability and makes it less scalable. Neu-NoC [7] implements a hybrid topology between mesh and ring. It consists of local rings and a global mesh that interconnects all these rings. Neu-NoC consists of two types of routers based on wormhole flow control: a ring router connected to each PE in each ring and a mesh router connected to each local ring. These routers support multicast transmission. The two issues with the Neu-NoC architecture are the latency and limited bandwidth of the ring topology. MAERI [8] is a DNN accelerator built with a set of modular and configurable blocks that can easily support different DNN implementations. A fat-tree topology is used for data distribution as it is most suitable for multicast transfers. MAERI can be seen as a design methodology rather than a fixed architecture. The architecture can be adapted for specific DNN applications. MAERI offers greater flexibility, but the implementation of such architecture is complex and silicon area consuming.

The presented solutions use different approaches for their interconnect networks to support efficient data communications. These approaches attempt to optimise data transfers to improve performance and reduce power consumption. There is a clear trade-off between these optimisations and the flexibility required in terms of scalability and the ability to efficiently handle the data transfers resulting from the new DNN algorithms. Indeed, most of the studied architectures are designed to accelerate 2D convolutions of standard dimensions such as 3×3 or 5×5 . These architectures tend to connect these elements through a mesh network (Mesh) to take advantage of spatial parallelism and employ different dataflow models to exploit the notion of data reuse, which directly impacts the energy efficiency of the whole system.

A study of the on-chip communication properties in DNN accelerators has been done in [9]. In this study, different architectural configurations are evaluated to highlight features to be included in future on-chip communication architectures for DNN accelerators. The results show that the dataflow architectures have to: (i) guarantee sufficient bandwidth not to compromise computation, (ii) be flexible and scalable to fit the variability of DNNs, and (iii) limit access to external memories to reduce power consumption. Based on these characteristics, we propose the AINoC network that provides multiple data access

(broadcast/multicast) and high bandwidth to support parallel processing in DNN accelerators. AINoC provides data reuse to reduce memory access and power consumption. It has a configurable architecture to facilitate the mapping of different CNN layers of varying sizes, allowing the required adaptability on the dataflow and scalability to a large number of PEs.

3 AINoC architecture

AINoC, as shown in Fig. 1, is composed of a set of homogeneous routing devices interconnected through their four ports to form a 2D interconnect network grid providing communication in the different directions: North, East, West and South. Each router is connected via a local port to a processing node (N) through a network interface. Each router has a couple of addresses (i,j) , representing its position in the AINoC network with respect to a row X and a column Y.

This section will present the architecture of the routing devices and the features of AINoC designed to support flexible communication with many non-blocking multicast/broadcast data transfers. The AINoC execution model based on data reuse and communication/ computation overlap will also be detailed.

3.1 Parallel Routing Device

The parallel routing is ensured, as shown in Fig. 1, at the level of the different functions of the router device: buffering, control, arbitration and switching, to guarantee a large bandwidth and flexible communication. Indeed, through several buffering modules (e.g. FIFO, First-In-First-Out), different communication requests received in parallel can be stored without any loss. These requests are then processed simultaneously in several control modules. These modules ensure a deterministic control of the data transfer according to a static X-Y (X-direction priority) routing algorithm and management of different communications (unicast, multicast and broadcast). Parallel arbitration of the processing order of incoming data packets, according to the Round-Robin Arbitration (RRA) [10] based on scheduled access, provides better collision management, i.e., a request that has just been granted will have the lowest priority on the next arbitration cycle. Parallel switching comes next to simultaneously route data to the right outputs according to the Wormhole switching [11], i.e., the connection between one of the inputs and one of the outputs of a router is maintained until all the elementary data of a message packet are sent and this in a simultaneous way through the different switching modules.

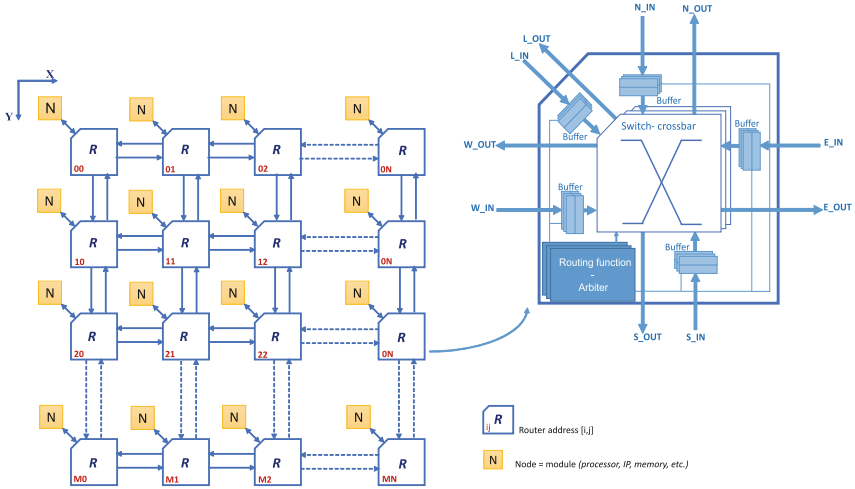


Fig. 1. AINoC architecture

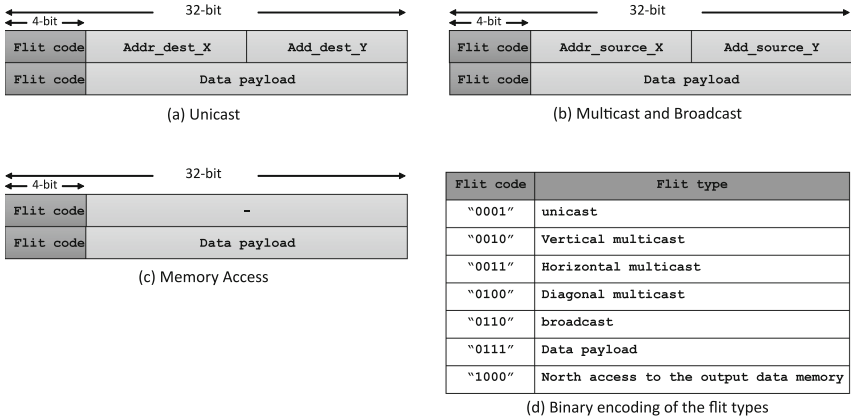


Fig. 2. Packet format for (a) unicast, (b) multicast/broadcast and (c) memory access, and (d) binary encoding of the flit types

The data packet format is shown in Fig. 2. The package is of variable size and is composed of two flits: a control flit and a payload flit. The packet flit consists of a header (flit code) followed by a data flit. The size of the flit depends on the size of the interconnect network since as the number of routing devices increases, more bits are needed to encode the addresses of the receivers or senders. Similarly, the flit size varies with the size of the payloads (filter weights, activation inputs or partial sums) to be passed through the network. In this paper, we have chosen to set the flit size to 32-bit to unify the configuration of the AINoC architecture. The value of the header determines the communication to be provided by the router. There are 4 possible types of communication: unicast, multicast

(horizontal, vertical, and diagonal), broadcast and memory access. The routing device first receives the control flit containing the type of communication and the source or destination address. The routing device decodes this control word and then allocates the communication path to transmit the payload flit that arrives at the cycle following the control flit. Once the payload packet is transmitted, the allocated path will be released for further transfers.

3.2 AINoC Features

A dedicated interconnect network must meet certain characteristics to be integrated into a CNN accelerator, such as (i) the capacity to deliver a massive amount of data (bandwidth) at low latency, (ii) the possibility to be flexible and scalable enough to adapt to the different shapes of the CNN layers, as well as (iii) the ability to limit the accesses to external memories to reduce the power consumption. The AINoC architecture is equipped with hardware features to guarantee a high-performance data exchange in a CNN accelerator to ensure these characteristics.

Multicast/Broadcast Without Retransfer. This mechanism is triggered during multicast and broadcast communications to avoid transfer re-looping and better control the data transmission delay throughout the network. Indeed, during a broadcast, packets coming from one or more directions will be transmitted to the other directions while the source direction(s) will be inhibited. This makes the maximum broadcast delay in a network of size $N * M$ equal to $[(N - 1) + (M - 1)]$.

Independent Communication Management. To guarantee and facilitate the overlap of communication with computation, a communication controller separate from the computation controller is used in AINoC (Fig. 1). Indeed, the computation controller controls the multiplication and accumulation operations and the read and write operations of the local memories (e.g. register file). In contrast, the communication controller manages the data transfers between the global memory and the local memories, and the inter-computation node data transfers. Synchronisation points between the two controllers are set up to prevent data overwriting or loss. With this communication control mechanism independent of the computation mechanism, we can ensure the transfer of weights in parallel with the transfer of ifmaps and the execution of communication operations in parallel with the computation. Thus, we can cover the communication by the computation and a communication by another communication.

3.3 Dataflow Model

An execution model of interconnect network must define the general organization of data exchanges between routers, processors and memories, without compromising the performance of the CNN accelerator.

Data Reuse Traffic. During the computation of the CNNs, several MAC operations are implemented, requiring multiple accesses to the same data. Since usually this data is stored in a separate memory (e.g. external DRAM or SRAM global buffer), repeated loading of this data is required, thus inducing excessive memory accesses. As a result, the power consumption due to memory accesses can be much higher than logical computations (e.g. MAC operation). To address this problem, reducing the number of memory accesses, e.g. by exploiting data reuse, can significantly reduce the power consumption of CNN accelerators.

AINoC supports several types of communication (unicast, vertical multicast, horizontal multicast, diagonal multicast and broadcast). This flexibility in the choice of the data transfer direction allows the reuse of all types of data in different directions. In this paper, we focus on the Row Stationary (RS) Dataflow model [3] which allows the reuse of three types of data: weights, input-feature-maps (ifmaps) and partial sums (psums).

To ensure better reuse of its data, an optimisation strategy has been defined to select the correct set of PEs to use among the available PEs of the hardware architecture. The dimensions of the set of PEs (and thus the target PEs) to be exploited are determined by the size of the filter and the output data (output-feature-maps, ofmaps) of a given layer. The use of this principle allows better management of energy consumption by reducing the cost of data transfer since it is optimised by the reuse of these data between the processing elements.

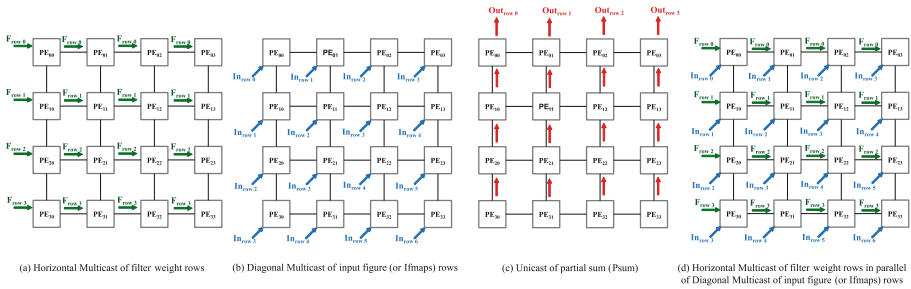


Fig. 3. Data reuse traffic

Figure 3 shows how data is reused in the network. For simplicity, we present the AINoC node as PE. Indeed, the rows of a single filter are reused horizontally across the PEs (a multicast of the filter weights Fig. 3(a)), the rows of input are reused diagonally across the PEs (a multicast of the input- image or ifmaps Fig. 3(b)), and the partial sums are accumulated vertically across the PEs (a unicast of the Psum Fig. 3(c)). AINoC can also perform parallel data-reuse of both filter weights and input-image/ifmaps, as shown in Fig. 3(d).

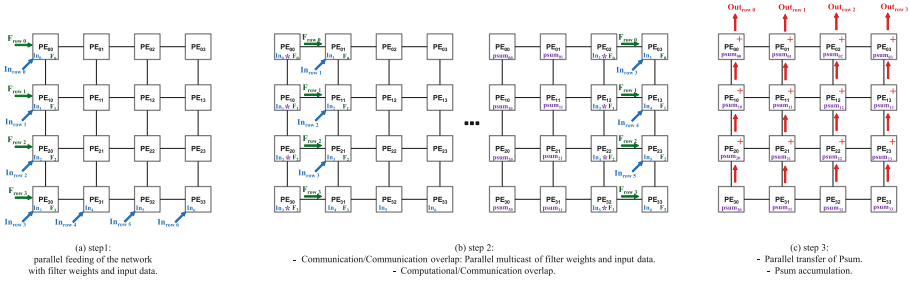


Fig. 4. Communication/computation overlap

Communication/Computation Overlap. It is a strategy for reducing the cost of data transfer to improve the execution time of parallel programs to reduce the effective contribution of the time dedicated to data transfer to the execution time of the entire application. The idea is to decouple computation and communication so that the computational units perform computational work while the communication infrastructure performs data transfer. This allows hiding partially or all of the communication overhead, knowing that the recovery can only be perfect if the computation time exceeds the communication time and the hardware allows to support this paradigm. To ensure a total or at least partial overlap, the communication control has been separated from the computation control while keeping synchronisation points between the two processes to facilitate their simultaneous execution.

Figure 4 shows how the recovery of communications is ensured by the computation and the realisation of different communications in parallel (the multicast of filter weights in parallel to the multicast of input data).

4 Experiment Results

This section is dedicated to obtaining and analysing the mapping results of different data flows generated by the different state-of-the-art CNN algorithms onto different hardware configurations of the AINoC-based dataflow architecture. The objective of these experiments is to evaluate the performance and characterise the AINoC network with respect to the number of routers and the interconnect bandwidth.

4.1 Evaluation Methodology

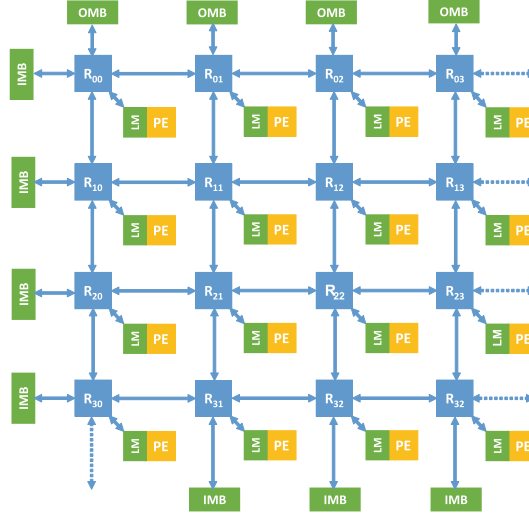


Fig. 5. AINoC configuration with filter size 4×4 and ifmap size 7×7 . IMB = Input Memory Buffer, OMB = Output Memory Buffer, R = Router, PE = Processing Element, LM = Local Memory

In this work, different CNN algorithms from state-of-the-art were used as case studies. They have different sizes and include different types of layers and shapes. LeNet5 [1] and MobileNetV2 [12] were chosen to have a collection of data resulting from a range of small to large CNN and using a set of layers including classical 2D convolution (CONV2D) and fully connected layers (FC) but also point-wise (PW) and depth-wise (DW) convolution layers in MobileNetV2. In our experimental study, we chose to test the key convolution layers that emphasise different filter sizes and ifmaps and the fully connected layers that require a linear spatial representation of the AINoC architecture. However, row width for FC layer (i.e., 1000) of MobileNetV2 is too big for AINoC-based dataflow architecture, so this layer has been excluded from our experiments because it does not fit on the target FPGA. We also note that an AINoC configuration must be generated according to the Eq. (1) for each evaluated layer to respect the RS dataflow execution mode, as shown in Table 1.

$$\begin{aligned} ofmaps_height = & (((ifmaps_height - (filter_height \\ & + padding_start + padding_end))/stride) + 1) \end{aligned} \quad (1)$$

Figure 5 shows an example of arrangement of AINoC of dimension 4×4 . Ofmap (stored in OMBs) size of 7×7 is determined from Eq. (1). Input data (filters and ifmaps) in each IMB is arranged by placing control words and payload words, where each control word is followed by a payload word. Prior to

execution, each payload word is stored in their respective LMs by routing them across the AINoC with the help of control words.

DROP acronym is used to represent the AINoC features, particularly, (1) Data Reuse, (2) Overlap, and (3) Parallel communication.

Table 1. CNN layers type. CH = CHannel, R = Row, C = Column

CNNs	Layer Type	ifmap size CH \times R \times C	filter shape CH \times R \times C	AINoC config. R \times C
LeNet5	conv_1	1 \times 32 \times 32	1 \times 5 \times 5	5 \times 28
	conv_2	6 \times 14 \times 14	6 \times 5 \times 5	5 \times 10
	conv_3	16 \times 5 \times 5	16 \times 1 \times 1	1 \times 5
	fc_1	1 \times 1 \times 120	1 \times 120 \times 84	1 \times 84
	fc_2	1 \times 1 \times 84	1 \times 84 \times 10	1 \times 10
MobileNetV2	conv_1	1 \times 128 \times 128	8 \times 3 \times 3 \times 3	3 \times 126
	conv_2	8 \times 64 \times 64	8 \times 3 \times 3	3 \times 62
	conv_3	24 \times 64 \times 64	24 \times 3 \times 3	3 \times 62
	conv_4	36 \times 32 \times 32	36 \times 3 \times 3	3 \times 30
	conv_5	48 \times 16 \times 16	48 \times 3 \times 3	3 \times 14
	conv_6	96 \times 8 \times 8	96 \times 3 \times 3	3 \times 6
	conv_7	144 \times 8 \times 8	144 \times 3 \times 3	3 \times 6
	conv_8	240 \times 4 \times 4	240 \times 3 \times 3	3 \times 2
	conv_9	80 \times 4 \times 4	256 \times 1 \times 1	1 \times 4

4.2 FPGA Implementation Results

The evaluation platform used for all tests is the Versal ACAP VCK190 kit [13] featuring an “XCVC1902-2VSVA2197” FPGA partition containing 899,840 programmable LUTs, 899,840 Flip-Flops, 1,968 DSP58, and 158 Mb of URAM and BRAM. The software tools used to implement and test the different AINoC configurations are:

- QuestaSim or Questa Advanced Simulator (version 2021.4) from Mentor Graphic is provided to simulate and test the programming and debugging of FPGA chips.
- Vivado Design Suite (version 2021.2) is a software suite produced by Xilinx to place-and-route and analyse hardware description language (HDL) designs.

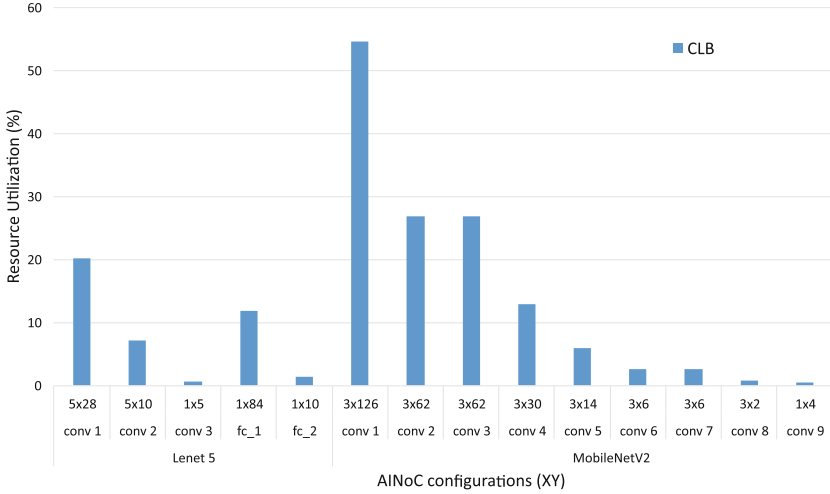


Fig. 6. Synthesis results of different AINoC configurations based on the CNN layer shapes

Area. To ensure the processing of different convolution and fully connected layers of the LeNet5 and MobileNetV2 networks, a CNN dataflow accelerator must be able to adapt its configuration according to the executed layer to optimise the consumption of the available hardware resources of the target FPGA. The AINoC interconnect network must also adapt its configuration to facilitate the scalability and flexibility of the accelerator.

Based on Table 1, which represents the different configurations of AINoC according to the shape of the layers of the chosen CNNs, we show the synthesis results of these different configurations in Fig. 6.

We note that larger size of the ifmaps results in larger size of the AINoC network. This is expected since the width of the network will depend on the lines of the ifmaps that will be stored in the memories connected to the routers of the last line of the network design (as shown in Fig. 5).

Throughput and Bandwidth Requirement. Figure 7 shows the throughput, bandwidth, and latency of AINoC while executing different layers of LeNet5 and MobileNetV2. These communications can be grouped into two, i.e., (1) parallel communication where data is being communicated from two sides of the AINoC and (2) single path routing where data is sent from only one side of the AINoC. Each group is then sub-grouped into four different communication types, i.e., (1) unicast, (2) vertical multicast, (3) horizontal multicast, and (4) broadcast. It can be observed that AINoC can provide a maximum throughput of 384 flits/cycle with packed data, i.e., $3 * 8$ -bit data in a 32-bit wide datapath and a maximum bandwidth of 51.2 GB/s for the conv_1 layer of MobileNetV2 because AINoC is connected to 128 input memory buffers in this configuration and providing maximum parallel access to memory buffers.

Latency. Figure 8 shows the latency performance of AINoC-based dataflow architecture while executing different layers of LeNet5 and MobileNetV2. It can be observed that AINoC can provide a gain of up to 71.74× (conv_1 of LeNet5) using the data reuse feature with respect to a single RISC-V CPU [14]. For latency comparison of AINoC-based dataflow architecture with RISC-V CPU, time required for access L2 to load data into IMBs (L1) is also considered for fair comparison.

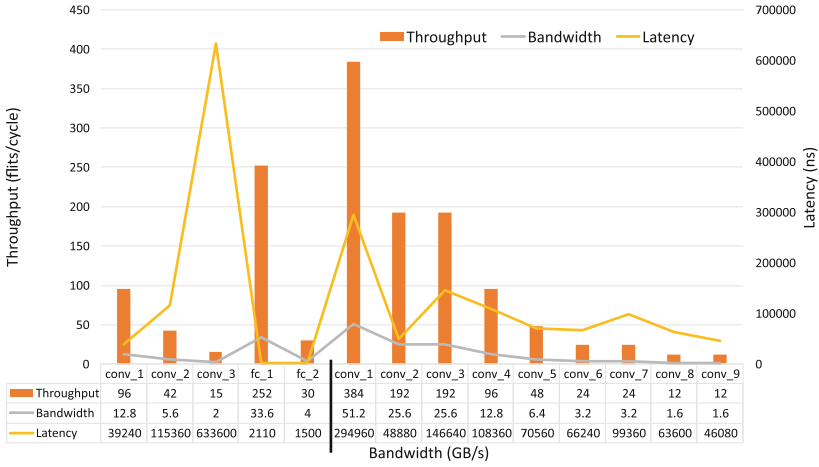


Fig. 7. Throughput (*flits/cycle*) vs. Bandwidth (*GB/s*) vs. Latency (*ns*) of different AINoC configurations based on the CNN layer shapes

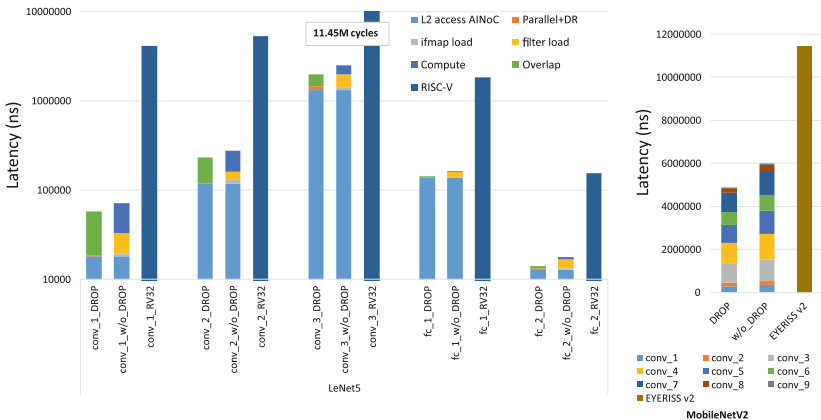


Fig. 8. Breakdown of latency (*ns*) results of AINoC-based dataflow architecture featuring with and without DROP for routing input data of convolution and fully-connected layers of LeNet5 and MobileNetV2

Overall speedup of MobileNetV2 convolution layers is up to $2.35\times$ w.r.t. Eyeriss v2 [3]. Here, Eyeriss v2 is executing all layers of MobileNetV2 while AINoC-based dataflow architecture is executing convolution layers only (Table 1).

In AINoC, the dimension and type of data routing significantly impact the latency performance of executing convolution layers. For LeNet5, the filter size for conv_1 and conv_2 is 5×5 , so during horizontal multicast, data routing is much better than the data routing of filters for layers of MobileNetV2, where filter size is 3×3 . Contrarily, conv_3 (LeNet5), conv_9 (MobileNetV2), and fully connected layers feature single row configurations of AINoC, and all routers are directly connected to the input buffers. It can be observed that AINoC-based dataflow architecture accelerates by up to $10.5\times$ (fc_1, LeNet5) using the DROP features with respect to a configuration of AINoC-based dataflow architecture without support for DROP because in such configuration, all routers in AINoC fetches the input data simultaneously, and there is no need for data reuse. This peculiarity of AINoC influences its energy consumption, and energy consumption section discusses such trade-off.

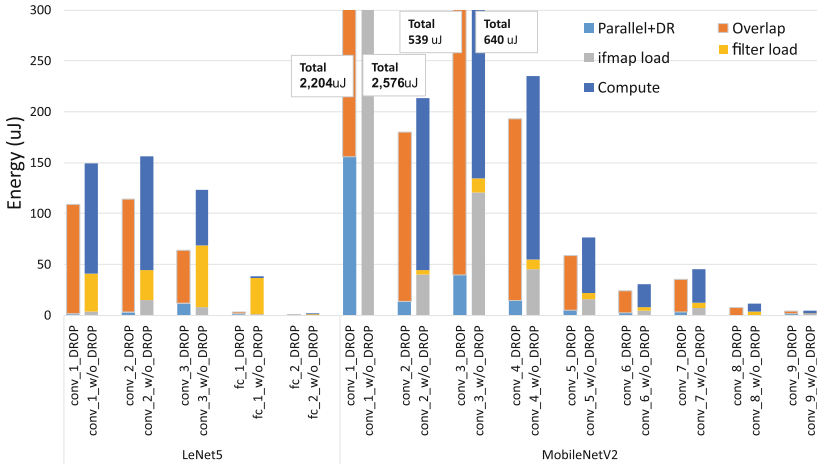


Fig. 9. Breakdown of energy consumption (μJ) of AINoC-based dataflow architecture featuring with and without DROP for routing input data of convolution and fully-connected layers of LeNet5 and MobileNetV2

Energy Consumption. The energy consumption of AINoC is different during memory access and data reuse. As mentioned in latency section, the AINoC configuration and type of data routing scheme influence its performance in executing convolution layers. Notably, during horizontal multicast of filter data, the number of parallel memory accesses is limited to the number of memory buffers directly connected to the first column of AINoC (as shown in Fig. 5). There is a trade-off between latency and energy consumption for routing filter data in AINoC, i.e., in the case of horizontal multicast, less number of parallel memory

accesses provide more room for data reuse and such execution period can be exploited to overlap the computation part of the convolution layer to better the energy efficiency of AINoC.

Figure 9 shows the energy consumption of AINoC-based dataflow architecture while executing different layers of LeNet5 and MobileNetV2. It can be observed that AINoC-based dataflow architecture reduces energy consumption by up to $10.9\times$ (fc_1, LeNet5) using the DROP features with respect to a configuration of AINoC-based dataflow architecture without support for DROP.

Due to different design flows i.e., Eyeriss v2 [3] is ASIC and AINoC-based dataflow architecture is FPGA, it is not a fair comparison between two architectures, and also due unavailability of design flow scripts for RISC-V CPU [14], we concluded to exclude the energy consumption comparisons for both architectures with AINoC-based dataflow architecture.

5 Conclusion

We presented a new interconnect network called AINoC, which is flexible to be configured for each CNN layer and consumes less energy to facilitate its integration into AI edge systems. We evaluated AINoC results using LeNet5 and MobileNetV2 to show their adaptability to a wide range of systems. AINoC is implemented (place and route) onto Versal ACAP VCK190 kit featuring XCVC1902-2VSVA2197 FPGA partition. While executing real world applications, AINoC can provide a maximum throughput of 384 flits/cycle with packed data and a maximum bandwidth of 51.2 GB/s for the conv_1 layer of MobileNetV2. AINoC by using DROP can speed up LeNet5 convolution layers by up to $71.74\times$ in latency performance w.r.t. a RISC-V based CPU and also speedup MobileNetV2 convolution layers by up to $2.35\times$ in latency performance w.r.t. Eyeriss v2. We plan, in the future, to optimise its performance and make the most of AINoC for accelerating complete Convolution Neural Networks execution.

Acknowledgements. The works have been funded by ECSEL-JU under the program ECSEL-Innovation Actions-2018 (ECSEL-IA) for research project CPS4EU (ID-826276) in the area Cyber-Physical Systems.




References

1. Lecun, Y., et al.: Gradient-based learning applied to document recognition. Proc. IEEE **86**(11), 2278–2324 (1998). <https://doi.org/10.1109/5.726791>
2. Brown, T.B., et al.: Language models are few-shot learners. arXiv (2020). <https://doi.org/10.48550/ARXIV.2005.14165>
3. Chen, Y.-H., et al.: Eyeriss: a spatial architecture for energy-efficient dataflow for convolutional neural networks. In: 2016 ACM/IEEE 43rd ISCA, pp. 367–379 (2016). <https://doi.org/10.1109/ISCA.2016.40>

4. Chen, T., et al.: DianNao: a small-footprint high-throughput accelerator for ubiquitous machine-learning. *SIGARCH Comput. Archit. News* **42**(1), 269–284 (2014). <https://doi.org/10.1145/2654822.2541967>
5. Du, Z., et al.: ShiDianNao: shifting vision processing closer to the sensor. in: 2015 ACM/IEEE 42nd ISCA, pp. 92–104 (2015). <https://doi.org/10.1145/2749469.2750389>
6. Luo, T., et al.: DaDianNao: a neural network supercomputer. *IEEE Trans. Comput.* **66**(1), 73–88 (2017). <https://doi.org/10.1109/TC.2016.2574353>
7. Liu, X., et al.: Neu-NoC: a high-efficient interconnection network for accelerated neuromorphic systems. In: 2018 23rd ASP-DAC, pp. 141–146 (2018). <https://doi.org/10.1109/ASPDAC.2018.8297296>
8. Kwon, H., et al.: MAERI: enabling flexible dataflow mapping over DNN accelerators via reconfigurable interconnects. In: Proceedings of the Twenty-Third ASP-LOS 2018, pp. 461–475. Association for Computing Machinery, New York (2018). <https://doi.org/10.1145/3173162.3173176>
9. Krichene, H., Philippe, J.-M.: Analysis of on-chip communication properties in accelerator architectures for deep neural networks. In: 2021 15th IEEE/ACM NOCS, pp. 9–14 (2021)
10. Shin, E.S., et al.: Round-robin arbiter design and generation. In: Proceedings of the 15th ISSS 2002, pp. 243–248. Association for Computing Machinery, New York (2002). <https://doi.org/10.1145/581199.581253>
11. Ni, L.M., McKinley, P.K.: A survey of wormhole routing techniques in direct networks. *Computer* **26**(2), 62–76 (1993). <https://doi.org/10.1109/2.191995>
12. Sandler, M., et al.: MobileNetV2: inverted residuals and linear bottlenecks. In: The IEEE CVPR, Salt Lake City, UT, USA, pp. 4510–4520 (2018). <https://doi.org/10.1109/CVPR.2018.00474>
13. Xilinx VCK190 Evaluation Board. User Guide UG1366 (v1.1) (2022). <https://docs.xilinx.com/r/en-US/ug1366-vck190-eval-bd>. Accessed 17 Oct 2022
14. Pullini, A., Rossi, D., Loi, I., Tagliavini, G., Benini, L.: Mr.Wolf: an energy-precision scalable parallel ultra low power SoC for IoT edge processing. *IEEE J. Solid-State Circ.* **54**(7), 1970–1981 (2019). <https://doi.org/10.1109/JSSC.2019.2912307>



Real-Time FPGA Implementation of the Semi-global Matching Stereo Vision Algorithm for a 4K/UHD Video Stream

Mariusz Grabowski  and Tomasz Kryjak  

Embedded Vision Systems Group, Computer Vision Laboratory,
Department of Automatic Control and Robotics,
AGH University of Science and Technology, Krakow, Poland
grabowski@student.agh.edu.pl, tomasz.kryjak@agh.edu.pl

Abstract. In this paper, we propose a real-time FPGA implementation of the Semi-Global Matching (SGM) stereo vision algorithm. The designed module supports a 4K/Ultra HD (3840 × 2160 pixels @ 30 frames per second) video stream in a 4 pixel per clock (ppc) format and a 64-pixel disparity range. The baseline SGM implementation had to be modified to process pixels in the 4ppc format and meet the timing constraints, however, our version provides results comparable to the original design. The solution has been positively evaluated on the Xilinx VC707 development board with a Virtex-7 FPGA device.

Keywords: SGM · FPGA · 4K · Ultra HD · real-time processing · stereo vision

1 Introduction

Information on the 3D structure (depth) of a scene is very important in many robotic systems, including self-driving cars and unmanned aerial vehicles (UAVs), as it is used in object detection and navigation modules. The depth map can be estimated using several different approaches, active: LiDAR (Light Detection and Ranging), Time of Flight (ToF) cameras, stereo vision with structured lighting; and passive: stereo vision. Stereo vision uses two or more cameras that acquire the same scene, but from slightly different points in space. A detailed discussion of the advantages and disadvantages of different sensors can be found in the work of Jamwal, Jindal, and Singh [1].

Stereo vision, in its passive variant, is an often used solution in embedded systems due to the low price of the equipment, its small size and weight (no need for a laser light source, rotating elements or projectors). The accuracy of the results obtained with this technology strictly depends on the algorithm used to process the acquired images. The methods used can be divided into two groups: local and global [2]. In both cases, the key element is to find the same pixels in the image captured by the left (usually considered as the base) and

right camera (reference). Their offset expressed in pixels is referred to as the disparity. This value can be easily converted to the distance from the sensors using the vision system parameters.

Global methods introduce appropriate discontinuity penalties in order to smooth the disparity map. Their aim is to optimise the energy function defined over the whole image. By means of global algorithms, much more reliable and accurate disparity maps are determined, but the smoothing task is NP-hard and algorithms are very computationally demanding, and for this reason they are not suitable for implementation in real-time systems.

It should be also noted that the current dominant trend is depth estimation using deep neural networks [3]. However, due to the high computational complexity, especially for high-resolution video streams, this topic remains outside the focus of our present work.

The SGM (Semi-Global Matching) algorithm was introduced by Hirsh Müller in 2005 [4] and 2008 [5]. It is based on two components: (1) matching at a single pixel level with the use of mutual information and (2) approximation of a global, two-dimensional smoothness constraint (obtained by combining multiple 1D constraints). The SGM algorithm is an example of an intermediate method between local and global approaches for determining disparity maps and is a compromise between accuracy and computational complexity. However, using SGM for high-resolution images is still challenging. For example, for a resolution of 1920×1080 pixels at 30 frames per second, an execution of about 2 TOPS (Tera Operations Per Second) with memory bandwidth of 39 Tb/s is required to process all pixels (2 million) [6].

In this paper we present an architecture of a stereo vision system with a modified SGM algorithm to process a 4K/Ultra HD (3840×2160 pixels @ 30 frames per second) video stream in 4ppc (pixel per clock) format and its implementation in an FPGA (Field Programmable Gate Array) device. The proposed modification solves the data dependency problem while not affecting the algorithm's accuracy. To the authors' knowledge, this is the only verified hardware implementation of the SGM method for 4K/Ultra HD resolution.

The remainder of this paper is organised as follows. In Sect. 2 we present basic information about the SGM algorithm. In Sect. 3 we review the previous work on SGM implementation on FPGAs. We describe the proposed method and architecture, as well as the evaluation of the algorithm and the hardware implementation in Sect. 4. The paper ends with conclusions and future research directions.

2 The SGM Algorithm

As mentioned in the introduction, the SGM algorithm is an intermediate approach between local and global methods for determining disparity maps. Furthermore, it is possible to implement it in an FPGA, in a pipelined vision system.

The input to the algorithm is a pair of rectified images. It consists of the following steps: calculation of the matching cost, aggregation of the cost

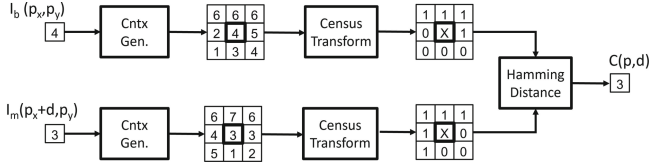


Fig. 1. Matching cost calculation with the Census transform and the Hamming distance metric, with example values.

(calculation of the smoothness constraint) and determination of the final disparity map.

Determining the correspondence between pixels using only the matching cost alone can lead to ambiguous and incorrect results. Therefore, an additional global condition is proposed in the SGM algorithm, which adds a “penalty” for changing the disparity value (i.e., supports the smoothness of the image), by aggregating the costs along independent paths.

In this work, the cost of matching $C(p, d)$ between a pixel $\mathbf{p} = [p_x, p_y]^T$ from the base image I_b , and the potentially corresponding pixel (shifted by the disparity d in a horizontal line) in the reference image I_m , is calculated using the Census transform and the Hamming distance measure, as shown in Fig. 1.

Let $L_{\mathbf{r}}$ denote the path in the direction \mathbf{r} . The path cost $L_{\mathbf{r}}(\mathbf{p}, d)$ is defined recursively as:

$$\begin{aligned}
 L_{\mathbf{r}}(\mathbf{p}, d) = & C(\mathbf{p}, d) + \min[L_{\mathbf{r}}(\mathbf{p} - \mathbf{r}, d), \\
 & L_{\mathbf{r}}(\mathbf{p} - \mathbf{r}, d - 1) + P_1, \\
 & L_{\mathbf{r}}(\mathbf{p} - \mathbf{r}, d + 1) + P_1, \\
 & \min_i L_{\mathbf{r}}(\mathbf{p} - \mathbf{r}, i) + P_2] \\
 & - \min_k L_{\mathbf{r}}(\mathbf{p} - \mathbf{r}, k)
 \end{aligned} \tag{1}$$

where: $C(p, d)$ is the matching cost, and the second part of the equation is the minimum path cost for the previous pixel $\mathbf{p} - \mathbf{r}$ on the path, taking into account the corresponding discontinuity penalty. Two penalties were applied in the algorithm, P_1 for a 1-level change in disparity and P_2 for a larger change.

Finally, the matching cost is given as:

$$S(\mathbf{p}, d) = \sum_{\mathbf{r}} L_{\mathbf{r}}(\mathbf{p}, d) \tag{2}$$

The author of SGM recommend aggregation along at least 8 paths, i.e, vertically, horizontally and diagonally in both directions (cf. Fig. 3), although he suggests that good results are achieved for the number 16. The disparity map D_b corresponding to the base image I_b is obtained by selecting for each value \mathbf{p} the disparity d that corresponds to the minimum cost i.e, $\min_d S(\mathbf{p}, d)$. Optional element of the algorithm is the final post-processing: median filtering and map consistency check (so called left-right consistency check).

Due to the reasonable trade-off between computational complexity and the quality of the resulting disparity map, the SGM algorithm has become very popular. It is a basic method in the popular OpenCV library and the Computer Vision Toolbox of the Matlab software. It also provides an attractive solution for hardware implementations in FPGAs.

3 Previous Work

The topic of implementing stereo correspondence using FPGAs is very extensive, and hence this review is narrowed only to selected articles describing the SGM algorithm. Interested readers are referred to the review [7].

The paper written by Gehrig, Eberli, and Meye in 2009 [8] described an SGM architecture for processing images with a resolution of 750×480 pixels (effectively 340×200) @ 27 fps at 64 levels of disparity. It is worth noting that this was the first implementation of the SGM method in an FPGA.

The paper of Hofmann, Korinth, and Koch from 2016 [9] also proposes a hardware implementation of the SGM algorithm. The architecture features scalability and combines coarse-grain and fine-grain parallelisation capabilities. The authors performed tests for different configurations and resolutions. For 1920×1080 pixels @ 30 fps and 128 disparity levels, real-time processing was achieved at a clock of 130 MHz (VC709 board with Virtex-7 FPGA device).

In the paper of Zhao et al. from 2020 [10], the authors presented the FP-Stereo library, which uses the HLS language and allows the creation of SGM disparity calculation modules. The module has been designed in the form of an accelerator interfacing with a DMA controller, rather than directly with the video stream. For a 300 MHz clock, a resolution of 1242×374 pixels and 128 disparity range, 161 fps were achieved on the ZCU 102 board with the Xilinx Zynq UltraScale+ MPSoC device.

In the latest publications by Shrivastava et al. in 2020 [11] and Lee with Kim in 2021 [6], the support for parallel pixel processing has been added to increase throughput. In this approach, the main challenge is the presence of an inherent data dependency. In the paper from 2020 [11], it is addressed by *dependency relaxation*, i.e, the aggregation is performed on the basis of the pixel k earlier, where k is the number of pixels processed simultaneously. The author points out that such a solution represents a trade-off between accuracy and throughput.

In the work from 2021 [6], on the other hand, a different approach is presented, in which operations involving the inherent data dependency are performed not on a single pixel, but on a vector of pixels. This allows the generation of disparity maps with very close accuracy to the original SGM algorithm. In both solutions, the matching costs are determined based on the Census transform. In the first publication [11], for images at a resolution of 1280×960 pixels and disparity range of 64, 322 fps, and in the second [6] for a resolution of 1920×1080 pixels and disparity range of 128, 103 fps were obtained.

We also propose a solution to the inherent data dependency problem. Our architecture is based on estimating the previous pixel aggregation cost on a path

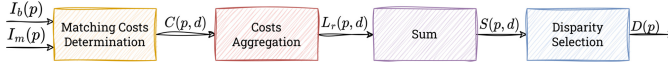


Fig. 2. A general scheme of the proposed SGM disparity estimation system.

with minimal additional logic needed. That allows us to process images with a 4K resolution and also to obtain comparable results to the original SGM algorithm without parallelism.

4 The Proposed Hardware Implementation

The aim of our work was to implement a hardware architecture capable of processing a video stream with a resolution of 3840×2160 pixels in real-time (i.e. processing 30 frames per second with no pixel dropping). That stream transmitted in a 1 pixel per clock format requires a pixel clock frequency of approximately 250 MHz. Adding to this value i.e. the vertical and horizontal blanking fields, the required clock equals about 300 MHz, which is too high for the rather complicated SGM algorithm. At the bottleneck, cost aggregation calculations take more than 10 ns on our platform. So, in order to process the data in the desired resolution, it is necessary to introduce parallelisation. In this work, a 4ppc (pixel per clock) format is used, in which 4 pixels are processed in parallel. This allows the pixel clock to be lowered to approximately 75 MHz. However, the use of such format has significant implications on the implementation of the SGM algorithm, due to the inherent data dependency.

A general scheme for the proposed vision system is shown in Fig. 2. The module accepts a synchronised video stream of rectified images, the base $I_B(p)$ and the reference $I_M(p)$ one. Further processing consists of several steps: determination of the matching cost $C(p, d)$ using the Census transform based matching method, calculation of the cost aggregation $L_r(p, d)$, summation of the aggregation costs from all directions $S(p, d)$ and disparity determination $D(p)$.

4.1 Determination of the Matching Cost

The 4ppc format does not introduce major complications into the hardware architecture of the matching cost determination module, but only increases the hardware resource requirements. First, 5×5 contexts are created for both images. For the base image, in a given cycle, 4 contexts are created (as implied by the 4ppc format [12]), and for the reference image this number is increased by the disparity range ($4 + \text{disp_range} - 1$), so that it is possible to simultaneously compare each of the 4 contexts of the base image with all the contexts in the disparity range of the reference image. A Census transform is performed on the generated contexts, and the contexts are then compared accordingly using the Hamming distance metric. The output consists of matching cost vectors.

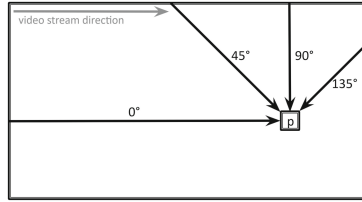


Fig. 3. Cost aggregation paths in SGM.

4.2 Cost Aggregation

In the next step, a quasi-global optimisation is performed by aggregating the costs for the whole image according to the SGM algorithm. In the current version of the module, this is implemented on four paths in the directions 0° , 45° , 90° , 135° , as shown in Fig. 3, which can be processed directly (without additional video stream buffering).

Theoretically, it is also possible to realise the other four directions (180° , 225° , 270° , 315°), but this would require storing the entire image in external RAM, using additional resources of the FPGA device, complex control logic and introducing additional latency in image processing.

In order to calculate the aggregation cost for a given pixel, it is necessary to know the value of the aggregation cost for the previous pixel on the path (cf. Eqs. (1) and (2)). For the 45° , 90° , 135° paths, the aggregation costs for the pixels in a given line are stored in Block RAM and read out accordingly during the processing of the next image line to calculate the costs for the subsequent pixels on these paths. The hardware architecture of this computation is shown in Fig. 4 and follows Eq. (1). The grey part is replicated for the entire range of disparities (*disp range*) and performs in parallel and one block of finding the minimum value of aggregation costs of the previous pixel on the path $\min L_r(p - r)$ is exploited to calculate the aggregation cost for the current pixel for each disparity value in the range.

For the 4ppc format, the difficulty arises for the 0° path. Using the aggregation cost of the previous pixel $L_r(p - r, d)$, which for this path lies in the same image line and potentially in the same 4ppc format data vector, results in the need to process four pixels in the same clock cycle. In the worst case, for the last pixel in the vector, in one clock cycle the data would have to propagate through four serially connected aggregation cost calculation units, as in Fig. 4. The critical path would contain 4 minimum modules of size *disp range*, four minimum modules of size 4 and 12 adders/subtractors. For this reason, the cost aggregation based on a baseline architecture (i.e., as proposed by the authors of SGM) for the 0° path is not feasible for the considered 4K resolution, without violating timing constraints.

It is therefore necessary to propose a new solution for the calculation of the aggregation cost for the 0° path. Time constraints require that the new architecture does not introduce significant additional propagation time and maintains

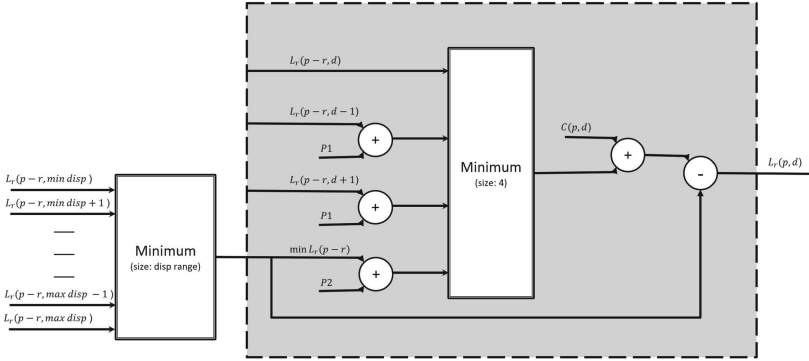


Fig. 4. Hardware architecture of the aggregation cost calculation unit for path r , pixel p and disparity d .

the approximation assumption of the global smoothness constraint of the SGM algorithm.

In our work, we designed and implemented an architecture with a proposed estimation of the aggregation cost value for consecutive pixels based on the calculated aggregation cost for the last pixel of the previous 4ppc vector (the pixel processed in the previous clock cycle) and the matching costs of the previous pixels in the same 4ppc vector.

For the first pixel in the 4ppc vector, the aggregation cost of the previous pixel is available during the calculation (it was calculated for the previous 4ppc vector), i.e:

$$L_r(p_1 - r, d) = L_r(p_{last}, d) \tag{3}$$

where: $L_r(p_1 - r, d)$ is the aggregation cost of the previous pixel relative to the first pixel in the 4ppc vector ($p_1 - r$), and $L_r(p_{last} - r, d)$ is the aggregation cost of the last pixel in the previous 4ppc vector.

For the consecutive pixels, we propose an estimation, which is performed according to the following Equations:

$$\begin{aligned} L'_r(p_2 - r, d) &= L_r(p_{last}, d) + \frac{1}{\lambda} (C(p_1, d) - L_r(p_{last}, d)) \\ L'_r(p_3 - r, d) &= L_r(p_{last}, d) + \frac{1}{\lambda} \left(\frac{C(p_1, d) + C(p_2, d)}{2} - L_r(p_{last}, d) \right) \\ L'_r(p_4 - r, d) &= L_r(p_{last}, d) + \frac{1}{\lambda} \left(\frac{\frac{C(p_1, d) + C(p_2, d)}{2} + C(p_3, d)}{2} - L_r(p_{last}, d) \right) \end{aligned} \tag{4}$$

where: $L'_r(p-r, d)$ is the estimated aggregation cost for the previous pixel relative to the pixel p , $C(p, d)$ is the matching cost for a given pixel, and the coefficient

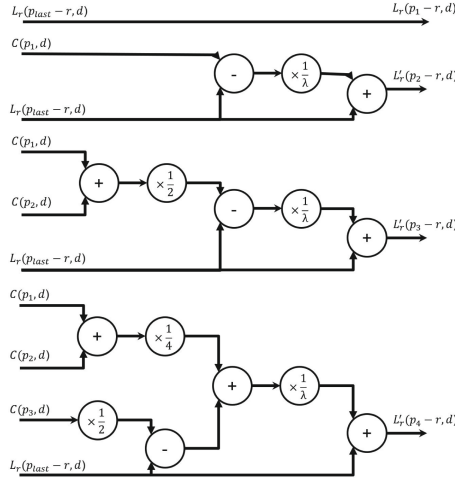


Fig. 5. The architecture for estimating the aggregation cost of the previous pixel for each pixel in the 4ppc vector.

λ may take a value which is a power of two (1, 2, 4, 8, 16, ...). The architecture of this solution is shown in Fig. 5.

The algorithm is based on the difference of the matching cost values of the previous pixels in a given 4ppc vector with the aggregation cost for the last pixel of the previous vector. The aggregation cost estimation architecture consists of basic components and introduces an additional delay only by the propagation time of the 3 adders/subtractors (critical path for $L_r(p_4 - r, d)$). Note: multiplication/division by a number that is a power of two is only a bit shift and requires no delay in the hardware implementation.

The solution takes into account the matching cost values of all previous pixels with the possibility to adjust the impact of the matching cost of previous pixels in a given vector by a factor of λ .

The estimated aggregation costs are then used to calculate the aggregation costs according to the architecture in Fig. 4. In the work of Shrivastava et al. [11] the estimation has been omitted and in the work of Lee and Kim [6] it has been solved by the cluster-wise cost aggregation.

The aggregation costs from all paths are then summed and the disparity is calculated. This involves finding the minimum matching cost.

4.3 Evaluation of the Proposed Method

The accuracy evaluation of the proposed algorithm was performed on a set of stereo images from the Middlebury 2014 [13] dataset. We skipped the final post-processing to better highlight the differences between the base SGM algorithm and the modified version proposed in this paper (SGM 4ppc). The accuracy was measured by the ratio of pixels with incorrect disparity value to all pixels of the

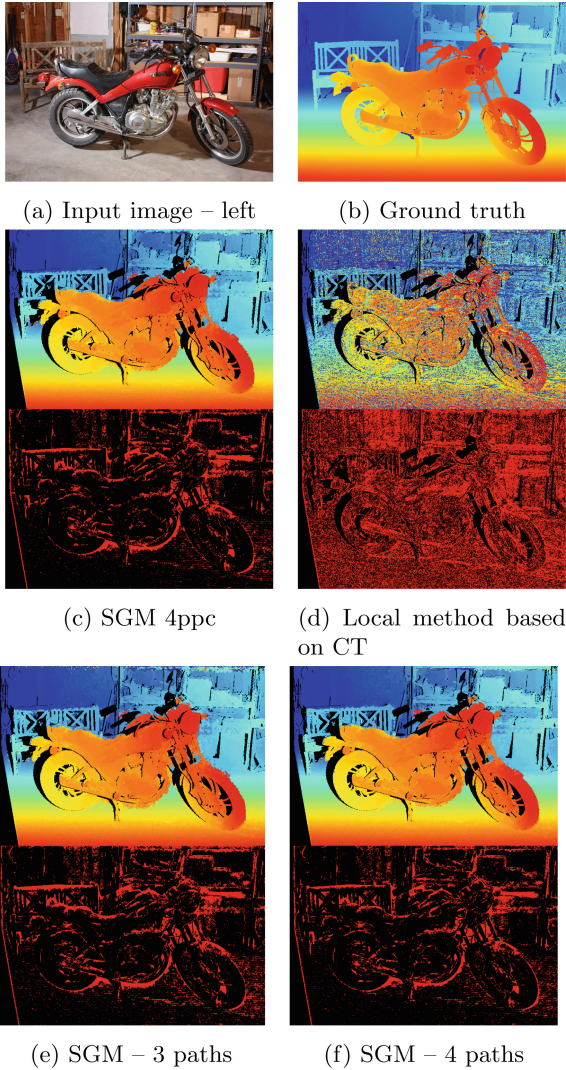


Fig. 6. Comparison of output disparity maps for the *Motorcycle* image in Middlebury 2014 dataset: (a) the left input image, (b) the ground truth disparity map, (c), (d), (e), (f) estimated disparity maps (on the top) and the error maps (on the bottom).

image (all) and also to the non-occluded (noc) pixels (occluded pixels should be filled with the Left/Right Check post-processing).

We compared the proposed method (SGM 4ppc) with the conventional local block matching based on the Census transform and the SGM algorithm (also based on the Census transform) with 3 and 4 aggregation paths. Figure 6 shows

Table 1. Comparison of error rates for the Middlebury 2014 dataset, based on all (all) and non-occluded (noc) pixels.

	all	noc
Local based on CT	68.21%	63,36%
SGM 3 paths	38.01%	28.79%
SGM 4 paths	36.27%	26.88%
SGM 8 paths	33.31%	23.11%
SGM 4ppc	36.64%	27.32%

sample evaluation results on the *Motorcycle* images from the Middlebury 2014 dataset. Table 1 shows the average evaluation results for the entire dataset.

The accuracy of the proposed method is comparable to the original SGM algorithm with 4 paths. The difference between error rates is about 0.4%.

4.4 Hardware Implementation

We implemented the proposed stereo vision system on a VC707 evaluation board with Xilinx’s Virtex-7 XC7VX485T-2FFG1761C device. We set up a test environment to evaluate the system, with test images sent directly from a PC to the board and later displayed on a 4K monitor.

We compared our solution with previous FPGA implementations of the SGM algorithm in Table 2. We used the following metrics: Frames per Second (FPS), Million Disparity Estimates per second (MDE/s) and MDE/s per Kilo LUTs (Look-Up Tables) (MDE/s/KLUT). First of all, our solution is the only one verified in hardware for a 4K/ Ultra HD resolution. We also would like to point out that the lower performance in FPS and MDE/s relative to previous work from 2020 [11] and 2021 [6] is due to the use of an FPGA chip with fewer resources. For this work, it was necessary to select a suitable platform to enable image acquisition in 4K resolution (i.e., having two high-bandwidth FMCs (FPGA Mezzanine Connectors) to which TB-FMCH-HDMI4K modules were attached).

It is also worth mentioning that the used FPGA technology differs not only in the number of resources but also in the performance. To compare: the critical path propagation time for the technology used in this paper after synthesis is 12.967 ns, but for the Xilinx Virtex UltraScale+ XCVU9P-L2FLGA2104E FPGA with the same parameters, it is 8.240 ns (36.45% faster).

Table 2. Comparison with previous FPGA implementations of the SGM algorithm.

	Image resolution	Disparity range	Platform	FPGA resources			Throughput		
				LUT	FF	BRAM	FPS	MDE/s	MDE/s/KLUT
[9]	1920 × 1080	128	Virtex-7	195k	217k	368	30	7 963	40.84
[14]	1600 × 1200	128	Stratix-V	222k	149k	N/A	43	10 472	47.2
[11]	1280 × 960	64	Virtex-7 690T	211k	N/A	641	322	25 056	118.6
[6]	1920 × 1080	128	Zynq US+	222k	135k	252	103	27 297	123.0
New	3840 × 2160	64	Virtex-7 485T	138k	65k	197	30	15 925	116.2

5 Conclusion

In this paper, we presented a hardware architecture for an SGM algorithm to process a 4K/Ultra HD video stream in real-time. We proposed a solution to the inherent data dependency problem. It allowed us to maintain high accuracy of the depth map estimation, while making it possible to take advantage of the 4ppc vector format. We implemented the module on a Virtex-7 FPGA platform achieving 30 frames per second for a resolution of 3840 × 2160 pixels with 64 disparity levels.

In future work, we plan to add more aggregation paths to the algorithm. With that, it will be possible to get more accurate results, but at the cost of latency and resource usage. We also plan to implement a video stream rectification module.

Acknowledgements. The work presented in this paper was supported by: the National Science Centre project no. 2016/23/D/ST6/01389 entitled “The development of computing resources organization in latest generation of heterogeneous reconfigurable devices enabling real-time processing of UHD/4K video stream”, the AGH University of Science and Technology project no. 16.16.120.773 and the program “Excellence initiative—research university” for the AGH University of Science and Technology.

References

1. Jamwal, N., Jindal, N., Singh, K.: A survey on depth map estimation strategies. In: International Conference on Signal Processing (ICSP 2016), pp. 1–5 (2016). <https://doi.org/10.1049/cp.2016.1453>
2. Scharstein, D., Szeliski, R.: A taxonomy and evaluation of dense two-frame stereo correspondence algorithms. *Int. J. Comput. Vision* **47**, 7–42 (2002). <https://doi.org/10.1023/A:1014573219977>
3. Laga, H., Jospin, L.V., Boussaid, F., Bennamoun, M.: A survey on deep learning techniques for stereo-based depth estimation. *IEEE Trans. Pattern Anal. Mach. Intell.* **44**(4), 1738–1764 (2022). <https://doi.org/10.1109/TPAMI.2020.3032602>
4. Hirschmuller, H.: Accurate and efficient stereo processing by semi-global matching and mutual information. In: 2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR 2005), vol. 2, pp. 807–814 (2005). <https://doi.org/10.1109/CVPR.2005.56>

5. Hirschmuller, H.: Stereo processing by semiglobal matching and mutual information. *IEEE Trans. Pattern Anal. Mach. Intell.* **30**(2), 328–341 (2008). ISSN 0162-8828
6. Lee, Y., Kim, H.: A high-throughput depth estimation processor for accurate semiglobal stereo matching using pipelined inter-pixel aggregation. *IEEE Trans. Circ. Syst. Video Technol.* **1** (2021). <https://doi.org/10.1109/TCSVT.2021.3061200>
7. Wan, Z., et al.: A survey of FPGA-based robotic computing. *IEEE Circ. Syst. Mag.* **21**(2), 48–74 (2021). <https://doi.org/10.1109/MCAS.2021.3071609>
8. Gehrig, S.K., Eberli, F., Meyer, T.: A real-time low-power stereo vision engine using semi-global matching. In: Fritz, M., Schiele, B., Piater, J.H. (eds.) *ICVS 2009*. LNCS, vol. 5815, pp. 134–143. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-04667-4_14. ISSN 03029743
9. Hofmann, J., Korinth, J., Koch, A.: A scalable high-performance hardware architecture for real-time stereo vision by semi-global matching. In: 2016 IEEE Conference on Computer Vision and Pattern Recognition Workshops (CVPRW), pp. 845–853 (2016). <https://doi.org/10.1109/CVPRW.2016.110>
10. Zhao, J., et al.: FP-Stereo: hardware-efficient stereo vision for embedded applications. In: 2020 30th International Conference on Field-Programmable Logic and Applications (FPL), pp. 269–276 (2020). <https://doi.org/10.1109/FPL50879.2020.00052>
11. Shrivastava, S., Choudhury, Z., Khandelwal, S., Purini, S.: FPGA accelerator for stereo vision using semi-global matching through dependency relaxation. In: 2020 30th International Conference on Field-Programmable Logic and Applications (FPL), pp. 304–309 (2020). <https://doi.org/10.1109/FPL50879.2020.00057>
12. Kowalczyk, M., Przewlocka, D., Kryjak, T.: Real-time implementation of contextual image processing operations for 4k video stream in Zynq ultrascale+ MPSoC. In: 2018 Conference on Design and Architectures for Signal and Image Processing (DASIP), pp. 37–42 (2018). <https://doi.org/10.1109/DASIP.2018.8597105>
13. Scharstein, D., et al.: High-resolution stereo datasets with subpixel-accurate ground truth. In: Jiang, X., Hornegger, J., Koch, R. (eds.) *GCPR 2014*. LNCS, vol. 8753, pp. 31–42. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-11752-2_3. ISBN 978-3-319-11752-2
14. Wang, W., Yan, J., Ningyi, X., Wang, Yu., Hsu, F.-H.: Real-time high-quality stereo vision system in FPGA. *IEEE Trans. Circuits Syst. Video Technol.* **25**(10), 1696–1708 (2015). <https://doi.org/10.1109/TCSVT.2015.2397196>



TaPaFuzz - An FPGA-Accelerated Framework for RISC-V IoT Graybox Fuzzing

Florian Meisel^(✉) , David Volz , Christoph Spang , Dat Tran ,
and Andreas Koch 

Embedded Systems and Applications Group (ESA), Technical University
of Darmstadt, Karolinenplatz 5, 64289 Darmstadt, Germany
{meisel, volz, spang, koch}@esa.tu-darmstadt.de

Abstract. Fuzz testing, which repeatedly executes a given program with auto-generated random inputs, and records its dynamic control flow, aims to discover sources of unexpected program behavior impacting security, which can then be fixed easier by directed developer effort. When targeting IoT devices, fuzzing faces the problem that the small IoT processors often lack the observability required for fuzzing, e.g., a high-performance trace port, while software-emulation on a faster host CPU is often slow, and compilation of the IoT application to a different ISA for faster native execution on the host introduces inaccuracies in the fuzzing process. To overcome all three of these drawbacks for RISC-V-based IoT processors, which are expected to dominate future IoT applications with their lack of ISA licensing costs, we modify an open-source RISC-V core for use in an FPGA-based hardware-accelerated fuzzing system. Our fuzzer has demonstrated up to *four times* the performance of the state-of-the-art QEMU-based fuzzing tool AFL++, even when running on very fast x86 host processors clocked at 4.95 GHz.

Keywords: Security · Fuzzing · LibAFL · TaPaSCo · RISC-V · Coverage

1 Introduction

With the global number of IoT devices continually rising, a single security vulnerability may result in thousands of affected devices at once [4]. To prevent attackers from quickly accumulating large nets of devices under their control, software security is key. Testing the device firmware for issues can help in detecting many potential weaknesses, but also increases development costs and is too often deemed infeasible. Automatically generating test cases by using a fuzzer framework is one way to effectively search for vulnerabilities in the firmware of such devices. A fuzzer can find vulnerabilities in a *target* program by repeatedly executing it using inputs from a generated *corpus* of inputs. The fuzzer traces

the Control Flow (CF) of the running program, detecting invalid program states (such as crashes, timeouts or memory leaks) in the process.

Fuzzing the firmware of a RISC-V IoT device on x86 hardware, however, comes with drawbacks, which are addressed in this work by modifying an existing RISC-V core to support fuzzing in hardware on FPGA:

Precision: Due to ISA differences, cross-compiled RISC-V program binaries differ from their x86 pendants. These differences originate for example from different instruction mappings, potentially changing addressing, word width, and compiler backend optimizations. Fuzzing a program in the host computer’s native ISA yields a chance of finding program bugs that would not actually apply to the RISC-V version. On the other hand, it might miss bugs that would occur only in the RISC-V ISA. Fuzzing the original RISC-V program binary thus has a better chance of precisely finding the relevant bugs.

Emulation Overhead: Emulating an ISA results in a huge runtime overhead. The AFL++ documentation estimates x2 to x5. We measured x20 overheads, and assume the difference to be due to the lack of advanced AFL++ features such as *persistent mode* on the RISC-V platform [2]. In contrast, native execution carries *no* emulation overhead.

Monitoring Overhead: Graybox software fuzzing frameworks implement their CF monitoring by patching additional function calls into the program to be tested (target), causing interrupts and overheads at runtime. As an alternative, monitoring can be implemented in hardware running in *parallel* to the actual software execution, ideally with no additional runtime overhead.

Our main contributions are

- Seamless integration of the FPGA accelerator into an existing software fuzzer framework. Its usage becomes as easy as using a plain ISA-emulating fuzzer.
- Compared to prior work, we rely on hardware extensions instead of software patches, allowing to fuzz the original program in real time. We contribute a new hardware unit, which is being attached to a RISC-V processor core for monitoring and compressing the target program’s CF events.
- An AXI wrapper for legalizing aborted AXI transfers, which would otherwise hang due to random partial design resets occurring between fuzzer job runs. As the wrapper operates solely on the AXI and reset interfaces, it is portable across RISC-V core microarchitectures and different AXI components.
- We contribute microarchitecture fixes to the CVA5 RISC-V core, allowing it to fully reset its caches and branch predictors between fuzzer runs.
- We reach up to 4.5x wall clock speedups over the traditional emulation-based methods in job launch rate, and an improvement of five additionally detected CF edges over one hour.

Sections 2 and 3 give background information and related work. Section 4 contains the implementation. Sections 5 and 6 evaluate and conclude.

2 Fundamentals

Fuzz-Testing is a well-established research area, this section can thus only cover fundamentals. For an overview of the current research, we recommend [11].

Definition - Fuzzing: A fuzzer is an application, which iteratively executes a test program with varying inputs. In literature, the program that is being tested is called a *target*. A fuzzer’s generation of new target inputs can be either generation- or mutation-based. The set of known input test cases is called *corpus*. For mutation-based fuzzers, new inputs mutate from initial *seeds*, which define the starting configuration. Both generation and mutation based fuzzers can be aware of the input’s legal structure, e.g., a JSON file, which is provided to the target. In addition to program inputs, awareness of the target’s internal structure and state helps in increasing the coverage. Beyond job execution rate, the rate of coverage growth is also influenced by the search strategy. Finally, fuzzers are categorized into black-, white-, and graybox fuzzers:

Black-, White-, and Graybox Fuzzers: Lacking a feedback loop for program-internal state, blackbox fuzzers are unaware of the target’s internal structure. They monitor external behavior such as crashes to evaluate the target’s state.

Whitebox fuzzers use static code analysis to direct a target’s CF towards higher coverage, or to focus on user-defined critical program regions.

In contrast to whitebox fuzzers, graybox fuzzers collect CF information via a feedback loop during target runtime. Typically, this is implemented by instrumenting (patching) the target, which causes significant memory and runtime overheads and may also alter the program’s behavior.

The **Graybox Fuzzer Result Aggregation (FRA)** may include different coverage information. First, basic block coverage provides information about which basic block (BB) has been executed. Additionally, the number of BB executions can be counted and visualized in a BB histogram. As an alternative, CF edge coverage tracks information on the actually taken CF edges.

Other coverage approaches are possible, but their benefit depends on the individual fuzzing target. For example, hash digests identifying entire CF paths guide towards high path coverage, which can be reached by mutating just one loop limit. It can find a new path per run, but miss other relevant CF edges.

3 Related Work

Quick EMUlator (QEMU) is a generic machine emulator and virtualizer [3]. QEMU executes non-native Instruction Set Architecture (ISA) programs by software emulation, exploiting dynamic translation to improve performance. For fuzzing, this enables us to fuzz-test software targeting IoT devices in their native ISA, rather than fuzz-testing in x86. All frameworks discussed in this section rely on QEMU to provide the capability of non-native ISA fuzzing.

American fuzzy lop (AFL) is a no-longer maintained open-source fuzzing framework developed by Michal Zalewski and later Google [16]. It contains tools

and fuzzing operation modes, and supports genetic algorithms for input mutation. AFL uses LLVM and GCC for target instrumentation and alternatively allows binary-only instrumentation. To fuzz non-instrumented targets, AFL will fall back onto a blackbox mode, and rely on crash and hang detection (timeout) for feedback. Additionally, AFL provides a QEMU mode to fuzz non-native ISA targets. The authors generally approximate QEMU mode’s runtime overhead between factors of 2x to 5x [2]. Beyond its powerful mutation algorithms, AFL is easy to use. To fuzz a target, a user provides the target and a dataset of one or more sample legal inputs. To optimize the re-spawning processes for fuzzer runs, AFL uses faster or a reduced number of fork system calls.

AFL++ is a community fork of AFL with a newer codebase and features [8]. AFL++ exposes a custom mutator API to enable researchers to implement plugins to combine new ideas with existing fuzzing technologies.

LibFuzzer is a fuzzing framework related to the LLVM project [14]. It is integrated into the target binary. The user provides an entry point to the target, from which libFuzzer spawns parallel threads to run with varying inputs. As limitations, the target may not modify global state or provide its own reset.

Real-Time: Some fuzzing techniques, e.g., used by AFL++ and libFuzzer, employ compiler transformations to make the application easier to fuzz. This ranges from instrumenting special tracing instructions to CF altering transformations. E.g., CF edges with complex conditions are hard to fuzz, because a specific edge is taken only when all partial conditions are met simultaneously. By splitting the condition over multiple basic blocks, the fuzzer receives more runtime feedback to find inputs that meet all partial conditions. As this transformation affects the runtime of the application, it may be inappropriate for real-time IoT targets.

4 Hardware/Software Co-designed Fuzzer

This section discusses our hardware/software co-designed fuzzer for RISC-V IoT firmware, and details hard- (Sect. 4.1) and software (Sect. 4.5) components.

Hardware: Our hardware component executes the IoT firmware program, captures the execution’s edge coverage map, and finally, together with the target’s return value, returns it back to the host software.

Software: In an iterative search, the host’s *fuzzer software* creates target inputs, launches the actual fuzzer runs, which traditionally would be executed in a RISC-V emulator, on the FPGA accelerator hardware instead, and finally evaluates the program execution to create the next iteration’s inputs.

4.1 Hardware - Interconnects (PE Ports)

For a seamless hardware/software integration, we employ the freely available Task Parallel System Composer (TaPaSCo) FPGA abstraction framework [9].

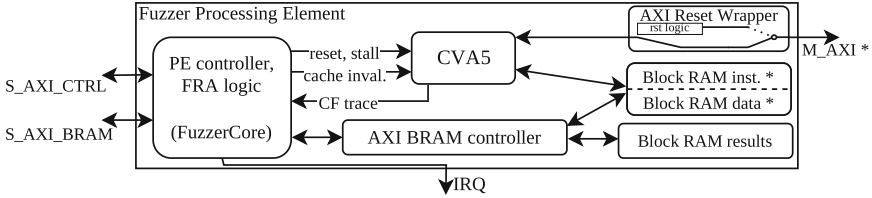


Fig. 1. Simplified layout of the Fuzzer TaPaSCo PE (based on [9,10]). The * marks different memory configurations.

TaPaSCo allows composing SoCs consisting of heterogeneous processing elements onto a wide spectrum of FPGA platforms. The tool also provides the drivers and middleware to communicate between soft- and hardware components.

As can be seen in Fig. 1, the TaPaFuzz Fuzzer Processing Element (PE), which contains the actual RISC-V core and FRA logic, has two AXI slave ports. One AXI port enables PE control, e.g., for reset and restart, and upon request, also provides status information. The second AXI slave port provides access to the PE’s memories, which are instruction, data, and the fuzzing result memory.

The PE has an optional AXI master port, which, when being connected to an AXI-DRAM controller, replaces or extends the BRAM-based instruction and data memories that are needed for larger, possibly non-IoT, targets.

Finally, a single-bit interrupt (IRQ) signals a finished (or broken) target execution to the host software. Next, we discuss the PE’s internals.

4.2 Hardware - Processor Core

A fuzzer PE consists of the CVA5 *RISC-V* processor [12,13], a job controller, and FRA hardware.

The PEs rely on CVA5 soft-core processors, which achieve a better performance than many other RISC-V cores when used on FPGA. Instead of a single execute stage, the CVA5 single-issue processor core has multiple independent functional units. This allows it to perform higher latency operations, such as memory loads/stores and divisions, without stalling, assuming that the immediately following instructions do not depend on their writeback results and can be handled by other currently idle functional units [13]. Furthermore, as an alternative to memory bus designs, CVA5’s optional BRAM instruction and data scratchpad interfaces considerably reduce memory access latency.

4.3 Hardware - Fuzzer Result Aggregation

General Mechanism. For reporting the fuzzing coverage, we implemented an edge coverage FRA (Sect. 2), which outputs a histogram of the taken CF edge transitions (the *fuzzer result map* or *coverage map*). In hardware, it is stored in a BRAM memory of configurable size, containing an 8-bit counter per edge.

While the RISC-V core runs the target software to be fuzzed (here: IoT firmware), the FRA hardware block receives the CF events. Branch and jump instructions in flight are detected by their instruction encoding. A hash from the instruction’s source and target Program Counters (PCs) is then generated and trimmed to an index into the coverage map, where the 8-bit counter corresponding to that control edge is then incremented. On the end of execution, the hardware signals an interrupt to the host, which in turn fetches the coverage map, exception status, and time (in cycles) to generate the next fuzzer run’s inputs.

Hashed Indexing of Control Flow Edge Counters. The FRA hardware is attached to the CVA5 core via the core’s built-in tracing interface, which provides dedicated PC and instruction word outputs. Based on each CF edge’s start and target addresses, the hash algorithm creates pseudo-random indices within the coverage map, whose values count each edge’s occurrences. The hash algorithm implementation needs to enable a high throughput, low risk of collisions, and low hardware overhead, while cryptographic security is not a requirement.

As SHA and similar hash algorithms do have massive hardware overheads and potentially a limited throughput, we decided on a suitable lightweight algorithm from the hash prospector repository [15]. It is not cryptographically secure, but runs with only 8 cycles of latency and guarantees a throughput of 1 item per cycle, thus does not limit the CF throughput. We feed the 32-bit hash algorithm with the edge’s source PC, add the destination PC to an intermediate value of the algorithm to avoid collisions with nearby CF, and trim the result’s bit-length depending on the chosen coverage map size to form a valid index.

Due to arbitration between PE-external and internal access, and due to the additional latency from the AXI BRAM controller, the overall jump and branch throughput is limited by the read-and-write round trip time to the result memory. A direct stall signal into the processor is triggered if required to not miss CFs.

4.4 Hardware Modifications for More Effective Fuzzing

The fuzzing use-case examined here has somewhat unusual requirements on the acceleration hardware due to the many resets that may occur when fuzzing discovers anomalous behavior, which is the goal of the entire fuzzing process. Thus, we need to enable the hardware to efficiently and reliably deal with these many resets. This requires extensions to the internal bus interfaces and, for the CVA5 core, to the cache and branch predictor.

Legalizing AXI Bursts in the Context of Partial Design Resets. When a RISC-V core is reset to prepare the next program execution, its internal bus component drops any ongoing transactions, while the external memory bus must remain active and is generally not able to deal with the abruptly aborted transfers.

Between individual executions, while the results are downloaded by the host and the program data memory is refreshed to its original state for the next fuzzing iteration, the fuzzing controller asserts the processor’s reset line to return it to a consistent state. As an alternative, it would be possible to keep the core active after a successful execution in a similar fashion to the persistent mode of *AFL++*. However, exceptions and especially timeouts would require an external control sequence to actually restart the program for the next iteration. That sequence would depend on the current processor state and the concrete application. This can be avoided by resetting the softcore processor subsystem.

With the BRAM resources being handled inside the PE, the core can be reset regardless of timing, as the BRAM interface does not have any handshakes or request sequences that need to be finished. However, the AXI4 specification [5] does not include any mechanism to safely abort its handshakes and transactions. As a consequence, for the fuzzer variant allowing access to external memories via an AXI bus, an arbitrary reset could lock up the entire design.

As a solution to this problem, we devise a *wrapper* module placed in-between the core and the downstream AXI4 memory bus to *complete* the remaining transactions even when the core resets, as shown in Fig. 1. While the core is operating normally, the wrapper combinationally passes through the AXI signals, but keeps a registered copy of each handshake from the core. In addition, it also maintains counters for in-flight requests and the remaining number of beats in a write transaction. The wrapper would only stall write handshakes in case the burst length FIFO or an in-flight access counter would overflow. Note that for the CVA5, no such stalling will occur in practice, as that core sends all write beats *before* starting the next burst.

On a synchronous local core reset, the wrapper module takes over the bus lines from the resetting core. First, pending handshake requests from the core are stabilized using the registered copies until accepted by the bus, if required, to match the AXI specification. Second, for each outstanding write burst, write beats with all bytes disabled ($wstrb \leftarrow 0$) are sent matching the burst length. Finally, the shim waits until all response handshakes (final read beats and write responses) arrive from the bus before notifying the fuzzing controller that the bus is now stable and the core reset can be released.

The shim module is intended to be portable across different cores or other AXI components by making only few assumptions beyond the AXI standard. These assumptions are that the core never issues handshakes for write beats (data) before the write burst request (address, length), and that no more than a configurable number of in-flight read requests (default 15) are sent by the core and accepted by the downstream bus.

Clearing CVA5 Cache Tags. If a different program is to be uploaded, or the data memory is to be restored to its original state, consistency with caches inside the core needs to be maintained. For the CVA5, the tag memories for the instruction and data caches, as well as the branch predictor, persist through a reset. Since reestablishing consistency after a program upload involves invalidating a

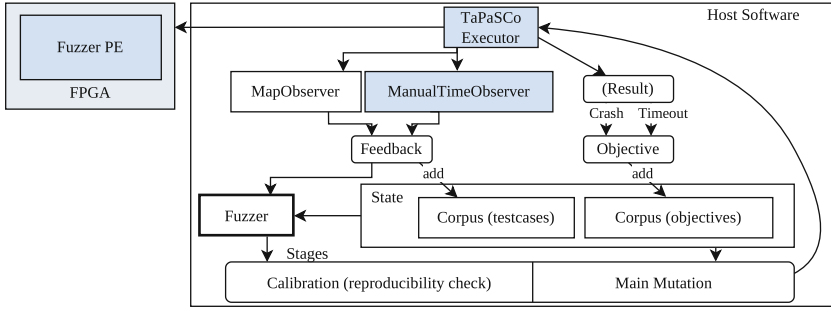


Fig. 2. Simplified layout of the fuzzer software, based on LibAFL’s StdFuzzer implementation [7]. Blue color denotes custom or customised components. (Color figure online)

significant portion of the set addresses in the processor’s caches, we implemented hardware support for accelerated invalidations. This allows to invalidate tags at the rate of one set per clock cycle, regardless of cache associativity.

4.5 Fuzzer Software Architecture

The software portion of our work uses LibAFL [6], an existing library for fuzzer development. Its main authors, Fioraldi and Maier, have also worked on the AFL++ [8] fuzzing engine. While being an independent project, LibAFL uses similar concepts and techniques to AFL++ and AFL, such as a *forkserver* for target execution and a variety of mutators [6, 8].

As a key difference, LibAFL provides abstract components for fuzzers but leaves open their concrete composition into an application. Aside from selecting fuzzer stages or mutators, the modular and abstract design of LibAFL also enables fuzzer developers to implement new components and, for instance, use alternative input data structures for target programs with existing modules. It is designed to minimize the need for library code forks in custom fuzzer development.

Fuzzer Components. A typical LibAFL fuzzer is constructed by instantiating interdependent modules (see Fig. 2), loading the initial corpus from disk and calling the *fuzzer* loop. The fuzzer loop, in turn, runs a fuzzer *strategy* consisting of one or several *stages* that, invoked with a *testcase* (program input previously deemed interesting) chosen by a scheduler, implement strategies to *mutate* inputs and running evaluations through an *executor* module.

This executor, which is the key contribution in our software, runs the program with given input data, captures runtime information such as the coverage *map*, and differentiates between normal runs, crashes and timeouts. Captured information is passed on via *observers* to a *feedback* function to determine whether the program inputs should be stored for future iterations. For instance, inputs that

uncover a previously unseen CF edge, or other notable coverage and program run time results, would ideally be detected as such and added to the *testcase corpus*.

The *objective* function is defined to detect erroneous behavior, including program crashes and excessive run time (timeouts), and determines whether the corresponding program input should be stored for manual analysis.

Execution Offloading Mechanism. For job execution on the FPGA PE, the *libtapasco* runtime library serves as a means to interact with the hardware.

The first step is to retrieve the PE object for the required Fuzzer PE. To do this, *libtapasco* internally queries a status core added during design composition, describing all available PEs and the composition details. The program and fuzzer result memory configuration for job dispatching is determined based on the PE type, as it differs between BRAM- and DRAM-based fuzzer configurations.

Our hardware-accelerated executor is, as are the existing executor modules in LibAFL, interfaced to by a single method call. Instead of locally running the program with inputs provided by the caller, it selects the PE, prepares and launches a job on it, waits for job completion, and retrieves the results.

Job preparation involves uploading the initial program data memory with inputs from the fuzzer engine, and also the program instruction memory on the first launch of the PE. For a PE using caches to speed up DRAM access, an explicit invalidation is requested for the data cache and, on the initial instruction memory upload, additional invalidations for the instruction and branch predictor caches. The PE controller is passed arguments via its Memory Mapped Input Output (MMIO) space, including the program input address range, the size of the fuzzer result map to create, the timeout cycle count, and the PE’s program memory DRAM address section.

On completion, the result and execution cycle count (i.e. execution time) fields are retrieved. The fuzzer result map is downloaded from the device. The executor returns a success, timeout, or crash status depending on the PE result, and passes the map and elapsed time on to the *Observer* objects in software.

5 Evaluation

In this section, we evaluate hardware overheads and full system wall clock execution time performance. Performance is compared to AFL++. Our FPGA designs are composed with TaPaSCo 2022.1 and Xilinx Vivado 2021.2 [9].

5.1 FPGA Design Resources

As shown in Table 1, compared to a plain CVA5 RISC-V core PE, the BRAM variant of this work uses 54%–66% additional LUTs and registers, respectively at a 10 MHz lower frequency. The alternative DRAM version of the fuzzer backed by DDR4 memory requires 2%–5% more registers and LUTs over the BRAM variant, but reduces the device BRAM resource footprint by 80 KiB, reflecting its tradeoff with additional cache and branch prediction logic enabled in the RISC-V processor, but elimination of scratchpad BRAM.

Table 1. FPGA utilisation (compared to total available) for the different PEs excluding auxiliary components, and clock frequency on the Alveo U280 device

Variant	BRAM, No Fuzzer HW	BRAM	DRAM via AXI4
LUT	4521 (0.35%)	6978 (0.54%)	7323 (0.56%)
Register	3467 (0.13%)	5745 (0.22%)	5877 (0.23%)
DSP	4 (0.04%)	10 (0.11%)	10 (0.11%)
BRAM	128 KiB (1.59%)	136 KiB (1.69%)	56 KiB (0.69%)
f_{\max}	400 MHz	390 MHz	360 MHz

5.2 Fuzzing Performance

We evaluated single-thread execution performance on the following systems:

FPGA system: Xilinx Alveo U280, PCIe 3.0 x16, AMD EPYC 7443P (4.0 GHz)

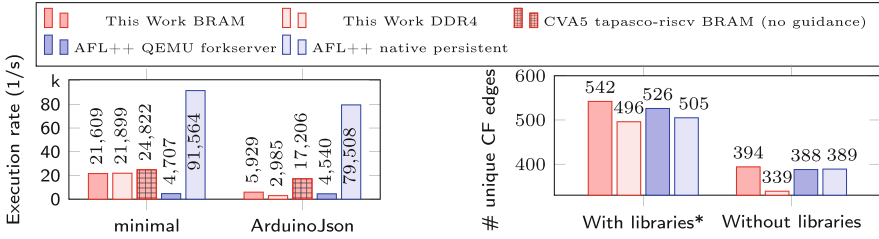
AFL++ evaluation system: AMD Ryzen 5900X (4.95 GHz)

We examine two target programs for this work. The first program, *minimal*, consists solely of the fuzzer-specific entry point calling an empty function; this serves as a peak execution rate benchmark. The second program, which we refer to as *ArduinoJson*, is a typical part of IoT applications that deserializes the input data using the *ArduinoJson* C++ library [1] into a dynamic memory buffer. It also makes limited use of floating-point arithmetic for number parsing.

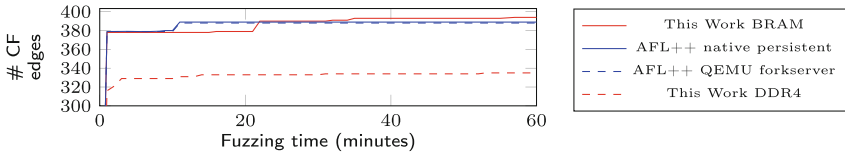
The programs are compiled in *-Os* mode, 1) for our RISC-V hardware fuzzer environment (*gcc*), 2) for a RISC-V Linux environment for AFL++ QEMU evaluation (*gcc*), and 3) with AFL++ native x86 instrumentation (*afl-clang-fast*). Note that the native x86 variant employs an AFL++ *persistent mode* harness to significantly reduce the number of *fork()* system calls, whereas the RISC-V Linux build uses a *forkserver* harness for the same purpose, since the faster AFL++ QEMU *persistent mode* is not supported for RISC-V targets [2].

Comparing the raw program execution rates in Fig. 3a, AFL++ in native persistent mode is significantly faster than both AFL++ QEMU forkserver mode and our work. But this approach carries the accuracy penalties discussed in Sect. 1. When performing the more accurate fuzzing on the actual RISC-V code, our work is 4.6x/31% faster compared to AFL++ QEMU when fuzzing the *minimal* and *ArduinoJson* programs, respectively. As we expected, runtime is increased when fuzzing the more complex *ArduinoJson* target. This is also because we chose larger coverage maps (2 KiB) that have to be transferred back to the host via PCIe, which impacts our speedup relative to AFL++ QEMU. The result for *ArduinoJson* without fuzzer hardware shows the impact.

Figure 3b and 3c show the absolute edge coverage attained after one hour of fuzzing. Notably, most edges are found within the first minute, indicating that the last edges are harder to find; the total number of reachable edges cannot be predicted. Since the FRA in our work includes all observed CFs, fuzzer guidance



(a) Comparison of the fuzzing execution rate with the *minimal* and *ArduinoJson* test programs between this work with a 64- (*minimal*) / 2048-entry coverage map (*ArduinoJson*, 1h), CVA5 without fuzzer hardware on a pregenerated testcase corpus, AFL++ with QEMU (RISC-V) and AFL++ native. (b) Comparison of the absolute edge coverage on the native RISC-V binary achieved after one hour of fuzzing the *ArduinoJson* test program, between this work with a 2048-entry map, AFL++ with QEMU and AFL++ native. *Without libraries* removes CF related to *libc* functions. *: For illustration only, as AFL++ guidance ignores libraries.



(c) Coverage over time without libraries, corresponding to Figure 3b. All lines start with 19 edges.

Fig. 3. Results: Fuzzer job execution rate and number of detected CF edges

also optimizes the library code coverage in contrast to AFL++. Overall, our BRAM fuzzer variant achieves the highest coverage both including and excluding library address ranges for *libc* and software floating point. The DRAM variant has the lowest result, which we attribute to lower execution rates from slower memory connectivity, due to which the fuzzer prefers inputs with simpler CF.

5.3 Hash Collisions

Since the coverage edges are hashed and then reduced to lower bit widths, collisions appear, such that two or more edges are assigned the same index in the map. Figure 4 quantifies this for the *ArduinoJson* example. If the overall cover-

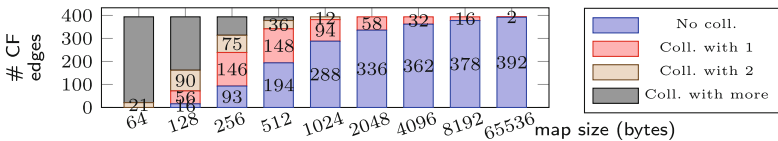


Fig. 4. Number of collisions for different fuzzer result map sizes, obtained via our hashing algorithm (Sect. 4.3) on the *ArduinoJson* corpus after 1 h of fuzzing.

age maps for inputs that reach certain colliding edges are similar, the fuzzer may store only one of the inputs in its corpus, leading to a suboptimal coverage. The probability of collisions drops with higher map sizes, which on the other hand increase the data transfer and processing overhead.

6 Conclusion and Future Work

In this work, we developed multiple enhancements for the CVA5 RISC-V core to make it more suitable for hardware accelerated fuzzing. The resulting solution is competitive when compared to an existing SoA fuzzer with software emulation, even when the latter employs a very fast desktop CPU as base for emulation.

For future work, we intend to scale the number of fuzzer units on the FPGA and optimize the DRAM fuzzer unit in order to compete with multithreaded fuzzing in software. Since the transfers of large coverage maps over PCIe currently limit performance, we will also explore approaches to reduce their required sizes.

Acknowledgements. The authors acknowledge the financial support by the Federal Ministry of Education and Research of Germany in the projects “Open6GHub” and “MANNHEIM-FlexKI” (grant numbers: 16KISK014, 01IS22086A-L). Part of this research work has been funded by the German Federal Ministry of Education and Research and the Hessian Ministry of Higher Education, Research, Science and the Arts within their joint support of the National Research Center for Applied Cybersecurity ATHENE.

References

1. ArduinoJson: Efficient JSON serialization for embedded C++. <https://arduinojson.org/>. Accessed 21 Oct 2022
2. High-performance binary-only instrumentation for AFL-fuzz. https://github.com/AFLplusplus/AFLplusplus/blob/stable/qemu_mode/README.md. Accessed 21 Oct 2022
3. QEMU - A generic and open source machine emulator and virtualizer. <https://www.qemu.org>. Accessed 05 Jan 2022
4. Clickguard - the most recent botnet attacks: The 2022 edition (2022). <https://www.clickguard.com/blog/recent-botnet-attacks-2022/>. Accessed 21 Oct 2022
5. Arm Limited: AMBA AXI and ACE Protocol Specification Version E, part 1. AMBA AXI3 and AXI4 Protocol Specification
6. Fioraldi, A., Maier, D.: The LibAFL fuzzing library. <https://afplusplus.com/libafl-book/libafl.html>. Accessed 17 July 2022
7. Fioraldi, A., Maier, D.: LibAFL StdFuzzer. <https://github.com/AFLplusplus/StdFuzzer>. Accessed 17 July 2022, commit c9f1bed
8. Fioraldi, A., Maier, D., Eißfeldt, H., Heuse, M.: AFL++: combining incremental steps of fuzzing research. In: 14th USENIX Workshop on Offensive Technologies (WOOT 20). USENIX Association (2020). <https://www.usenix.org/conference/woot20/presentation/fioraldi>

9. Heinz, C., Hofmann, J., Korinth, J., Sommer, L., Weber, L., Koch, A.: The TaPaSCo open-source toolflow. *J. Signal Process. Syst.* **93**, 545–563 (2021). <https://doi.org/10.1007/s11265-021-01640-8>
10. Heinz, C., Lavan, Y., Hofmann, J., Koch, A.: A catalog and in-hardware evaluation of open-source drop-in compatible RISC-V softcore processors. In: 2019 International Conference on ReConFigurable Computing and FPGAs (ReConFig), pp. 1–8 (2019). <https://doi.org/10.1109/ReConFig48160.2019.8994796>
11. Liang, H., Pei, X., Jia, X., Shen, W., Zhang, J.: Fuzzing: state of the art. *IEEE Trans. Reliab.* **67**(3), 1199–1218 (2018). <https://doi.org/10.1109/TR.2018.2834476>
12. Matthews, E., Shannon, L.: TAIGA: a new RISC-V soft-processor framework enabling high performance CPU architectural features. In: 2017 27th International Conference on Field Programmable Logic and Applications (FPL), pp. 1–4 (2017). <https://doi.org/10.23919/FPL.2017.8056766>
13. OpenHW Group: CORE-V CVA5 (repository). <https://github.com/openhwgroup/cva5>. Accessed 21 July 2022, commit 3239e20
14. The LLVM Project: libFuzzer - a library for coverage-guided fuzz testing. <https://llvm.org/docs/LibFuzzer.html>. Accessed 17 July 2022
15. Wellons, C.: Hash function prospector. <https://github.com/skeeto/hash-prospector>. Accessed 21 Oct 2022, commit 2051e59
16. Zalewski, M.: American fuzzy lop. <https://lcamtuf.coredump.cx/afl/>. Accessed 25 July 2022



Adaptive Inference for FPGA-Based 5G Automatic Modulation Classification

Daniel de Oliveira Rubiano, Guilherme Korol^(✉),
and Antonio Carlos Schneider Beck

Institute of Informatics - Universidade Federal do Rio Grande do Sul,
Porto Alegre, Brazil
{dorubiano,gskorol,caco}@inf.ufrgs.br

Abstract. Automatic Modulation Classification (AMC) is key to the efficient use of the radio frequency spectrum in modern applications, like 5G-based IoT. Optimizing AMC is crucial to achieving the latency, throughput, and energy levels expected by the final user. State-of-the-art solutions to the AMC problem are based on Deep Learning methods (e.g., Deep Neural Networks - DNNs). However, these methods require heavy processing and high energy consumption up to the point that accelerators (e.g., FPGA) are used to carry out such computations. Based on the observation that the classification becomes computationally harder or easier depending on the amount of noise the signal is subjected (i.e., Signal-to-Noise Ratio - SNR), this work proposes a fully adaptive FPGA-based inference system that selects the most appropriate DNN according to the current signal quality (SNR level). Compared to the state-of-the-art static approach, the framework reduces energy consumption by up to 43% while delivering $8.9\times$ more inferences per second.

Keywords: 5G Modulation · FPGA · Adaptive Inference · CNN

1 Introduction

Wireless networks, especially 5G, are the main enablers of the Internet of Things (IoT) revolution. However, researchers are challenged by the increase in the number of devices, data volume, and the complexity of these new 5G tasks. Even at the physical interface of 5G radio frequency (RF), heavy-load tasks (i.e., Deep Neural Networks - DNNs) are required to provide high-quality communication [12, 13]. In this context, the issue rises in aligning such computationally demanding tasks with the 5G user requirements of extremely low latency levels, high throughput, and, especially for the IoT, high energy efficiency.

In this work, we tackle an essential task on the 5G infrastructure: automatic modulation classification. Constant classification of the signal modulation is at the critical path of the RF interface of every 5G device. The modulation method is not defined by the communicating devices beforehand and, thus, must be classified, or detected, automatically and continuously [11]. One important aspect of

this task is the signal-to-noise ratio (SNR) to which the signal is subjected [19]. Higher noise situations make the classification task harder, requiring more complex (and costly) classifiers (e.g., deeper DNNs). On the other hand, a “cleaner” signal with high SNR makes the task easier, allowing for less complex classifiers. 5G, however, is highly dynamic with different environments and signal strengths that cannot be predicted at design-time. This creates room to optimize the modulation classification with respect to the current environment characteristics to improve latency, throughput, and energy aspects for the final user.

Therefore, to alleviate the high computational cost and efficiently demodulate signals, these DNNs can be offloaded to FPGA accelerators that will execute faster and at a better performance-energy trade-off than traditional CPU and GPU platforms in 5G base stations, Edge gateways, or even IoT devices [8]. FPGAs have been used with great success thanks to their reconfigurability, energy efficiency, and the emergence of High-Level Synthesis (or HLS) for easy FPGA programmability with high-level languages (e.g., C++).

In this scenario, we propose AIR-5G, a two-step framework for FPGA-based Addaptive DNN InfereRence for the modulation classification of 5G signals. First, at design-time, AIR-5G exploits the DNN hyper-parameters to generate multiple versions of DNNs and accelerators with different accuracy, performance and energy profiles. These versions are stored in a library for supporting the second, runtime, step. AIR-5G’s second step employs a selection algorithm to pick the most appropriate DNN/accelerator version from the library, taking into account current SNR level, a pre-defined optimization goal, and a minimum classification quality level (i.e., an accuracy threshold). With the right DNN/accelerator selected, AIR-5G can reconfigure the FPGA, adapting the inference processing to improve performance or reduce energy. In summary, the work makes the following contributions:

- It sets new frontiers in the design space of Deep Neural Network-based modulation classifiers on FPGA by exploring the accuracy-energy-performance trade-off;
- With the design space just created, and stored in the form of a library, AIR-5G adapts the inference processing at runtime according to current environmental conditions;
- Considering a 5G application scenario, our approach reduces energy consumption by up to 43%, while increasing inferences per second in up to $8.9\times$, when compared to a state-of-the-art solution.

2 Background and Related Work

2.1 Deep Learning for Radio Signal Classification

Traditionally, the problem of classifying RF signal modulation was addressed with specialized algorithms. For example, in [5] a likelihood approach is taken. Recently, Deep Learning (DL) and, in particular, Deep Neural Networks (DNN) have achieved state-of-the-art results in the automatic modulation classification

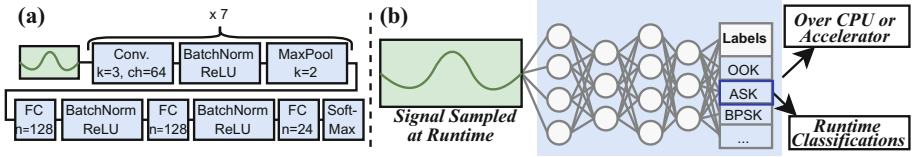


Fig. 1. Traditional approach for DL-based AMC.

(AMC) [12, 13]. In AMC, the classifier’s input is the sampled signal converted to the digital domain so that the carrier frequency is approximately aligned with the carrier of interest.

This paper discusses the use of Convolutional Neural Networks (CNNs) for AMC. The authors, O’Shea et al., have developed a state-of-the-art model based on the VGG-10 CNN (Fig. 1(a)), which is a widely used for image recognition. The model consists of seven convolutional layers (CONV of kernel size k and ch channels) with max-pooling and three fully connected (FC) layers of n neurons, and is trained to classify radio signals into one of 24 modulation classes. In [8], the authors propose a quantized version of this CNN, which reduces the model’s memory footprint and simplifies arithmetic operations, making it useful for constrained environments like 5G-based Internet of Things (IoT) systems [9, 10]. This quantized model uses 4-bit quantization for the weights and activations of all layers.

Figure 1(b) summarizes the traditional, *non-adaptive*, approach for AMC [2, 7, 12, 13, 17, 18]. This approach uses a single DL model that has been previously trained at design time considering a wide variation of SNR in the input signal. At runtime, the trained model runs on a CPU or dedicated accelerator to process (i.e., infer) new inputs.

2.2 FPGAs and Deep Neural Network Accelerators

The use of FPGAs as DNN accelerators has been mainly popularized by FPGA easy programmability (with frameworks for automatic mapping of DNNs), in addition to its reconfiguration capabilities. We highlight the FINN framework from Xilinx [1] that has been widely used in industry and academia and will be used in this work. FINN is an open-source tool that enables the construction of FPGA accelerators quickly and flexibly.

FINN is a tool that maps DNNs like the one in Fig. 1(a) into predefined High-Level Synthesis (HLS) modules. FINN allows for the adjustment of throughput (parallelism) and resource usage in each layer of the neural network through the mapping of the network and the parameterization of HLS modules. Parallelism is adjusted by setting the accelerator folding, which is defined by Processing Element (PE) and Single Instruction Multiple Data (SIMD) values for each module/layer through a JSON file.

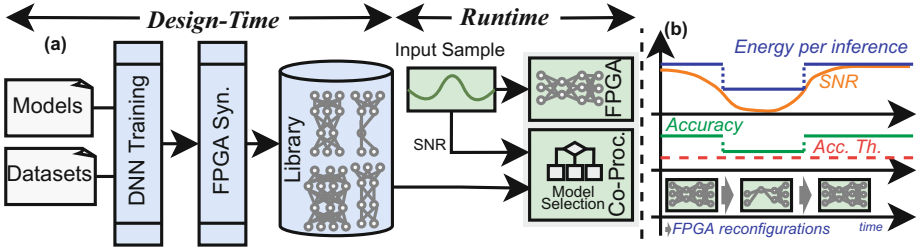


Fig. 2. (a) Overview of the AIR-5G framework and (b) its functioning explained.

3 System Overview

AIR-5G is a framework for FPGA-based adaptive inference for 5G AMC. Adapting the AMC for different SNRs makes it possible to obtain a better balance between energy, performance, and computational cost. Figure 2(a) illustrates our proposal. The solution is separated into two main steps: design-time and runtime. The targeted platform is a Xilinx ZCU104 MPSoC with FPGA (where the DNN inferences are executed) and a co-processor (where the runtime model selection runs). The remainder of the section details these steps.

3.1 Design-Time

Initially, at design-time, AIR-5G trains the multiple DNN models. Next, FPGA accelerator(s) synthesis of the trained models is carried out, generating a Library containing the multiple model/accelerator versions. At this phase, AIR-5G exploits the models hyper-parameters to create multiple versions, offering different computational costs, performance, and power profiles.

Training. In this step, AIR-5G performs the versioning, training and evaluation of the DNN models (*DNN Training* in Fig. 2(a)). For that, AIR-5G uses internally the PyTorch [15] and Brevitas [14] frameworks.

Training and evaluation start by reading the provided DNNs, such as the quantized version of the VGG-10 model described in Sect. 2.1. AIR-5G, then, progressively explores the DNN hyperparameters to create multiple versions from the same original model. The hyperparameters are the model quantization (bits for activation and weights), the number of channels in the convolutional layers, and the number of neurons in the fully-connected layers. By ranging the hyperparameters at fixed steps, model versions are generated and can be further trained on the provided dataset, evaluated, and exported as an Open Neural Network Exchange (ONNX) file to be further synthesized.

FPGA Synthesis. After training, the next step is synthesizing the FPGA accelerators for each model version (*FPGA Syn.* in Fig. 2(a)). All the model variants are synthesized with the same design parameters like clock period, timing constraints, and FINN configuration. Besides the FPGA synthesized bitstreams,

Table 1. Hyper-parameters used to generate the library.

<i>Model</i>	<i>Quantization</i>	<i>Convolutional Channels</i>	<i>Fully-Connected Neurons</i>
M1	2-bits	16	32
M2	2-bits	32	64
M3	2-bits	48	96
M4	2-bits	64	128
M5	4-bits	16	32
M6	4-bits	32	64
M7	4-bits	48	96
M8 (baseline)	4-bits	64	128

AIR-5G runs RTL simulations on the accelerators to assess performance metrics on each model. For synthesis and simulation, AIR-5G uses the FINN framework internally.

Finally, all generated accelerators are organized into a library supporting the runtime adaptability. The library is created with a table containing the accuracy of each model for all SNR values (obtained when evaluating the models in the DNN training step) and the design characteristics of the accelerators (e.g., power, throughput, resource utilization) gathered during the synthesis step.

3.2 Runtime

At runtime, the previously created library is used to adapt the inference processing according to the current SNR. To that end, the current SNR level is sent to the model selection module, running on the co-processor, which may reconfigure the FPGA with a new model/accelerator that best fits the current signal condition. The search happens at every new sampled SNR level.

Model Selection Module. The model selection algorithm runs on the embedded co-processor. It considers the current SNR value¹, the library produced at design-time, and two adjustable parameters defined prior deployment by the user: accuracy threshold and optimization criterion.

The accuracy threshold aims to adjust the maximum accuracy loss (w.r.t. the accuracy of the original model). The optimization criterion defines the model design characteristic that the algorithm must consider for selection. In this work, power and throughput criteria are used. However, we note that the algorithm can easily cover other metrics, such as latency or FPGA resource utilization.

Upon receiving the current SNR from the input sample, the model selection algorithm analyzes the accuracy under that particular SNR for the models in the library. It then filters out the models in which the drop in accuracy does not exceed the defined threshold. Thus, a subset of candidate models is formed.

¹ The SNR level can be safely profiled with, for instance, moment-based methods [4].

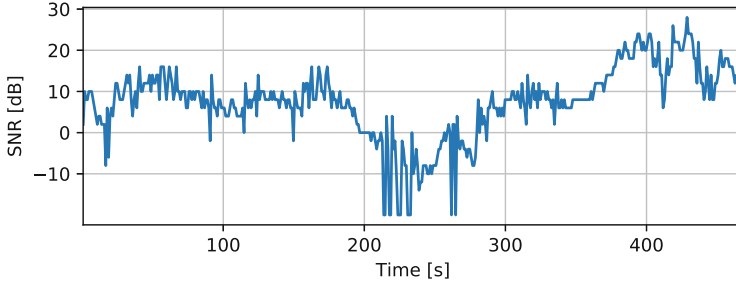


Fig. 3. SNR variation over the 468-second evaluation path.

Considering this subset, the decision for the most suitable model is made according to the optimization criterion. The algorithm analyzes the design data of the accelerators (e.g., power or throughput) and chooses the model that maximizes the defined criterion. For example, when defining the criterion for power optimization, the selected model will be the one with the lowest power dissipation among the subset of models with an accuracy loss below the threshold. Whenever the newly selected model diverges from the one currently loaded in the FPGA, the FPGA is reconfigured with the new model.

Figure 2(b) illustrates AIR-5G functioning. In that simple example, the changes in the SNR level (orange curve) caused two model switches, requiring two FPGA reconfigurations. At the first switch, we can see that, due to a poor SNR, AIR-5G loaded a simpler model (i.e., with fewer neurons) from its library to save energy (and also improve performance - not depicted) at the cost of some accuracy loss (which stays above the maximum allowed, accuracy threshold - red curve). As will be further detailed in Sect. 5, AIR-5G gains come from the fact that if no adaptation was done, the larger model, previously configured, would stay running on the FPGA - even though it would not keep its high accuracy because of the poor SNR level. In summary, AIR-5G leverages situations where high accuracy is unachievable to switch to simpler models, improving both energy and throughput aspects.

4 Methodology

Our 5G application case study is based on two publicly available datasets. The *RadioML 2018.01A* [13] dataset available at [3] was used to train and evaluate the CNN models. This dataset contains 2 million frames (2×1024 each), and are split as 90% for training and 10% for testing. The dataset offers SNR levels that can vary from -20 dB to $+30$ dB and must be classified among 24 modulation classes. Second, the *4G/LTE Bandwidth Logs* dataset [6] provides us with the measured quality of LTE connections recorded along different routes in a city while downloading a large file over HTTP (the connection quality is sampled at intervals from 700 to 1000 ms). The path chosen for evaluation was recorded on a moving vehicle and lasted 468 s. We used the Shannon-Hartley [16] theorem to

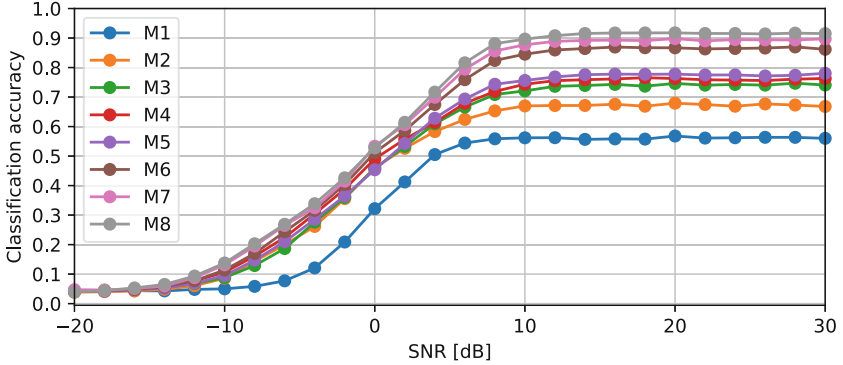


Fig. 4. Accuracy of the Library models w.r.t. SNR.

map the recorded signal quality to SNR values. Figure 3 shows the SNR variation over the 468-second path. Throughout the vehicle’s course, the SNR levels ranged from -20 to 28 dB (with a mean of 7.5 dB and mode of 10 db).

To create the Library, AIR-5G receives the baseline CNN [8], and reduces the number of convolutional channels and FC neurons at steps of 25% and quantization (for both activation and weights) of 2 and 4 bits. All models were trained for 20 epochs [8] with a batch size of 768 and a learning rate of 0.01. Then, the library models were synthesized within the FINN and Vivado tools from Xilinx at a frequency of 250 MHz targeting a ZCU104 board (xczu7ev). The FPGA power and resource utilization results were generated from Vivado and performance from RTL simulations with PyVerilator.

5 Results

We have split this section into two: first, we address how the heterogeneity created by multiple models and their synthesized accelerators translates into optimization opportunities. Later, at runtime, we show that AIR-5G can adapt the inference processing to the current SNR level.

5.1 AIR-5G at Design-time

Initially, AIR-5G created a library of eight distinct models, Table 1 presents each model version (from M1 to M8) hyperparameters. Figure 4 gives the accuracy (y-axis) of the models in the library over SNR values ranging from -20 to 30 dB (x-axis). Initially, we can verify that all models share a similar accuracy behavior of achieving better results as the signal quality improves. Moreover, that impact on accuracy is more significant in regions of SNR above 0 dB for all models in the library.

Now, let us detail the effects on the accuracy of quantization and model size (i.e., the number of CONV channels and FC neurons). For instance, take the

Table 2. Power and Performance and Resource Utilization for the Library Accelerators.

Model	Power (W)	Infer./s	Latency (cycles)	LUTs (%)	FFs (%)	BRAMs (%)
M1	0.716	238790	1688	07.10	04.72	00.32
M2	0.841	53753	15498	14.07	08.91	03.21
M3	0.891	35167	30545	14.75	09.66	07.21
M4	0.948	26141	52569	15.26	10.36	12.5
M5	0.876	238755	1676	19.86	09.00	00.66
M6	1.059	53752	15499	36.78	15.53	07.05
M7	1.245	35167	30546	37.03	16.82	11.22
M8 (baseline)	1.416	26141	52570	38.04	17.65	26.6

original model configuration (M8, gray curve) of 4-bit quantization, 64 channels in CONV layers, and 128 neurons in its FC layers. When comparing M8 to M4 (same size, 2-bit quantization, red curve) and M6 (same quantization, half of the original size, brown curve), we see that M6 stays closer to the baseline, indicating that the impact on accuracy is greater when reducing quantization compared to what is caused when the size (i.e., channels/neurons) is reduced.

More on that, when focusing on the region between +10 and +30 dB SNR, we can see the formation of two sets of models whose accuracy values have variation smaller than 10%. Both sets have in common the quantization, being one set formed by most of the 4-bit models (M6, M7, and M8), and a second set formed by models M3, M4, and M5 with lower accuracy levels. We can also see that M5, the smaller 4-bit model, which has 75% less CONV channels and FC neurons than M8 performs similarly to M4, the largest 2-bit model. It means that the impact on the accuracy of reducing in 75% the number of channels is equivalent to cutting in half the precision of weights and activations.

On the other hand, the region between -10 dB and 0 dB SNR shows a more compact distribution of the accuracy curves. In this case, the difference in accuracy between all models is at most 10%. When classifying samples at even lower SNR levels, as they approach -20 dB SNR, all models converge to an equally poor accuracy value.

Table 2 presents the power, throughput, and latency, as well as the FPGA resource usage of the accelerators synthesized from the trained models in the AIR-5G library (Table 1). As expected, resource usage and power increase as both the model size and the quantization increase. Following a similar analysis, we can compare pairs of models with the same number of channels/neurons (e.g., M8-M4, M7-M3) to see that throughput and latency are not impacted by quantization. This is because quantization does not interfere with the degree of parallelism of the accelerator: all models have the same topology (number of CONV layers and FC neurons), and all accelerators have the same pipeline stages configured in the same way (number of PEs and SIMD). On the other hand, quantization provides a considerable reduction in power, of approximately 33%, when comparing the baseline (M8) to its 2-bit counterpart (M4). Resource utilization follows a similar behavior.

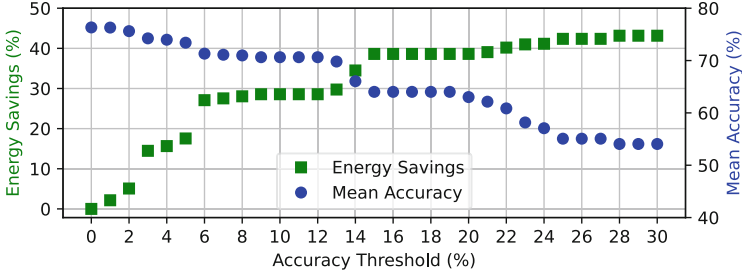


Fig. 5. Energy savings and mean accuracy over the baseline w.r.t. accuracy threshold.

As just discussed, the model versions offer an interesting set of possibilities to optimize the trade-off between accuracy, performance, and power over the SNR range. As we see next, such a trade-off can be exploited at runtime.

5.2 AIR-5G at Runtime

Energy. Figure 5 shows the energy savings (left y-axis) against the baseline (M8 in Table 1, statically deployed) for accuracy thresholds ranging from 0 to 30%, in increments of 1% (x-axis). Each point represents a different execution under a threshold on the same 468-second path with dynamic SNR level (see Fig. 3).

The first evaluated threshold is 0%, which is the baseline (i.e., original model, M8, without model switching). The highest evaluated accuracy threshold, 30%, grants the highest freedom for optimization, allowing a wider range of models to be selected (recall AIR-5G basic functioning from Fig. 2). In addition, Fig. 5 right y-axis presents the mean accuracy on each run. For this evaluation, the optimization criterion was set to power.

We notice that the reduction in energy consumption varies from approximately 2%, with the accuracy threshold at 1%, to a maximum of 43% with a threshold of 28%. Even at low threshold values, smaller than 5%, that allow a relatively small number of models from the library throughout the run, AIR-5G already has enough freedom to reduce energy consumption. This is because even models with a small accuracy drop already show some power reduction.

We can also notice a significant jump in energy savings when from 2 to 3% threshold, reaching 14.5%, while the mean accuracy suffers a reduction of only 2%. The same occurs between the 5% and 6% thresholds. A slight increase in the threshold at both points enables new models of lower power to be exploited at runtime. At threshold values above 20%, the gains in energy savings tend to grow smaller at each threshold increment. In such cases, the algorithm has a high degree of freedom of choice, tending to choose the least complex models in the ensemble (i.e. M1 and M2) at more points along the path. As the threshold approaches 30%, the algorithm tends to make more switches, prioritizing the model of lowest power dissipation. The preference for these models causes a steady drop in accuracy. Generally speaking, more flexible thresholds allow for a larger set of possible models that can, in turn, save more energy. When, however,

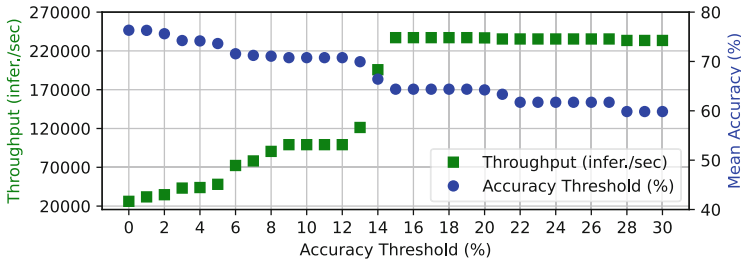


Fig. 6. AIR-5G average throughput (inferences/second) and mean accuracy.

the threshold does not allow new models to be selected (i.e., all models from the library have an accuracy drop below the threshold), energy savings and accuracy remain constant.

Performance. Similar to the results for energy, Fig. 6 presents the values of average throughput (left y-axis) and accuracy (right y-axis), for accuracy thresholds from 0 to 30% (each point evaluated under the same SNR variation - Fig. 3). For this evaluation, the optimization criterion was set to throughput. As a starting point, at 0% accuracy threshold, no accuracy drop is allowed and, thus, AIR-5G can only select model M8 (the baseline model) that delivers 26000 inferences per second (green curve, first marker on the left). Similar to what happens with energy, with small accuracy thresholds, AIR-5G is already capable of improving the inference processing. For example, the average throughput increases to approximately 43000 inferences per second with a threshold of 3%. The increases in throughput presented between the 13% and 15% thresholds are substantial, quickly reaching a plateau. From these thresholds onwards, AIR-5G is allowed to include the M5 and M1 models that are selected in intermediate and low SNR ranges, delivering the highest throughput levels.

For throughput, AIR-5G reaches a plateau at the 15% accuracy threshold. In the previous section, we saw that the accelerators throughput remains constant regarding quantization, where the 4-bit models have throughput values almost identical to the 2-bit models. At this point, the algorithm is allowed to choose models M7, M6 and M5, which have throughput values close to models M3, M2 and M1 (within 1% difference). Even with subsequent increments in the threshold, allowing models with 2-bit quantization to be selected, the throughput value is maintained. However, the accuracy levels keep falling since the system tends to select smaller models, not considering whether the increase in throughput is worth compared to their accuracy cost.

It is important to note that there is a slight decrease of 0.8% in the throughput values when the top thresholds are reached, between 27 and 30%. With such high thresholds, AIR-5G tends to select only the models with the highest throughput (M1 and M5) over different SNR values more frequently, causing more model switches and, consequently, more FPGA reconfigurations (70 in total, taking 10.15 s).

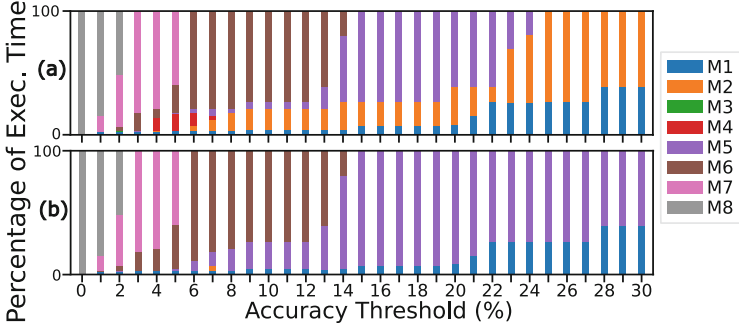


Fig. 7. Percentage of the execution time spent on each model version under power (a) and throughput (b) optimization criteria w.r.t. accuracy thresholds.

Adaptability. Figure 7 details the adaptation performed by AIR-5G. It shows the percentage of the execution time that each model version was selected during the full execution for each evaluated accuracy threshold. Upper plot presents the versions selected when the optimization criterion was set to power, while the lower plot presents the versions selected under the throughput optimization. Recall that under 0% threshold no version switch is performed and only the baseline model (M8) is used. We notice that as the accuracy threshold increases, different models get selected (i.e., poor accuracy models are only allowed at higher thresholds). Therefore, as larger accuracy drops are tolerated, the more costly versions are left unused (e.g., M8 gets selected up to 2% thresholds only).

As an example, let us take the 5% threshold in the upper plot. During this execution, five models were used (M1, M4-7). For that, 141 switches were performed, accounting for 4.36% of the execution time. Overall, we see that, despite interruptions due to FPGA reconfigurations, AIR-5G can provide better performance and energy efficiency than a statically deployed baseline (Figs. 5 and 6). Thanks to the models with varied performance, energy, and accuracy profiles in the AIR-5G library, AIR-5G can adapt the AMC at runtime through a runtime model selection algorithm according to the current environmental conditions.

6 Conclusion

We have presented an adaptive approach for DNN-based 5G AMC. Our framework can switch the DNN model (via FPGA reconfigurations) to best match the inference processing to the current signal quality. When compared to a statically deployed state-of-the-art accelerator, we achieve up to 43% reduction in energy and increase throughput up to $8.9\times$.

Acknowledgements. This study was financed in part by Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) - Brazil - Finance Code 001, São Paulo Research Foundation (FAPESP) grant #2021/06825-8, FAPERGS and CNPq.

References

1. Blott, M., et al.: FINN-R: an end-to-end deep-learning framework for fast exploration of quantized neural networks. *ACM Trans. Reconfigurable Technol. Syst.* **11**(3), 1–23 (2018)
2. den Boer, H., et al.: FPGA-based deep learning accelerator for RF applications. In: *MILCOM*, pp. 751–756. IEEE (2021)
3. Deepsig: Radioml datasets (2018). www.deepsig.ai/datasets
4. Ghodeswar, S., Poonacha, P.: An SNR estimation based adaptive hierarchical modulation classification method to recognize M-ary QAM and M-ary PSK signals. In: *ICSCN*, pp. 1–6 (2015)
5. Hameed, F., Dobre, O.A., Popescu, D.C.: On the likelihood-based approach to modulation classification. *IEEE Trans. Wireless Commun.* **8**(12), 5884–5892 (2009)
6. van der Hooft, J., et al.: HTTP/2-based adaptive streaming of HEVC video over 4G/LTE networks. *IEEE Commun. Lett.* **20**(11), 2177–2180 (2016)
7. Huang, L., et al.: Data augmentation for deep learning-based radio modulation classification. *IEEE Access* **8**, 1498–1506 (2020)
8. Jentzsch, F., et al.: RadioML meets FINN: Enabling future RF applications with FPGA streaming architectures. *IEEE Micro* **42**(6), 125–133 (2022)
9. Korol, G., et al.: Synergistically exploiting CNN pruning and HLS versioning for adaptive inference on multi-FPGAs at the edge. *ACM Trans. Embed. Comput. Syst.* **20**(5), 1–26 (2021)
10. Korol, G., et al.: AdaFlow: a framework for adaptive dataflow CNN acceleration on FPGAs. In: *DATE*, pp. 244–249. IEEE (2022)
11. Mitola, J., Maguire, G.Q.: Cognitive radio: making software radios more personal. *IEEE Wirel. Commun.* **6**, 13–18 (1999)
12. O’Shea, T.J., Corgan, J., Clancy, T.C.: Convolutional radio modulation recognition networks. In: Jayne, C., Iliadis, L. (eds.) *EANN 2016*. CCIS, vol. 629, pp. 213–226. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-44188-7_16
13. O’Shea, T.J., Roy, T., Clancy, T.C.: Over-the-air deep learning based radio signal classification. *IEEE J. Sel. Top. Signal Process.* **12**(1), 168–179 (2018)
14. Pappalardo, A.: Xilinx/brevitas (2021). <https://doi.org/10.5281/zenodo.3333552>
15. PyTorch: Pytorch (2022). <https://github.com/pytorch/pytorch>
16. Shannon, C.: Communication in the presence of noise. *Proc. IRE* **37**(1), 10–21 (1949)
17. Soltani, S., et al.: Real-time and embedded deep learning on FPGA for RF signal classification. In: *MILCOM*, pp. 1–6 (2019)
18. Tridgell, S., et al.: Real-time automatic modulation classification using RFSoc. In: *IPDPSW*, pp. 82–89 (2020)
19. Wu, Z., et al.: Robust automatic modulation classification under varying noise conditions. *IEEE Access* **5**, 19733–19741 (2017)



High-Level Online Power Monitoring of FPGA IP Based on Machine Learning

Majdi Richa¹(✉), Jean-Christophe Prévotet¹, Mickaël Dardailon¹,
Mohamad Mroué², and Abed Ellatif Samhat²

¹ Univ Rennes, INSA Rennes, CNRS, IETR - UMR 6164, 35000 Rennes, France
{[mricha](mailto:mricha@insa-rennes.fr), [jprevote](mailto:jprevote@insa-rennes.fr), [mdardail](mailto:mdardail@insa-rennes.fr)}@insa-rennes.fr
² Lebanese University, FoE, CRSI, Beirut, Lebanon
{[mohamad.mroue](mailto:mohamad.mroue@ul.edu.lb), [samhat](mailto:samhat@ul.edu.lb)}@ul.edu.lb

Abstract. Nowadays, power optimization has become a major interest for most digital hardware designers. Some, traditionally, might stick to offline power estimation especially in early design phases; some others resort to the modern and very promising runtime power management. Therefore, the Online Power Monitoring (OPM) is considered as an important feature serving real-time power optimization. OPM favors both in-situ power estimation and subsequent prediction, and can be exploited by the Dynamic Voltage and Frequency Scaling (DVFS) mechanism. DVFS, a modern technique for digital circuits power optimization, provides a realtime and adaptive voltage and/or frequency tuning while securing the systems' performance and integrity. In this paper, we present and evaluate an accurate online power monitoring methodology of FPGA IPs. We estimate power consumption using machine learning techniques, based on the IP's most power-influential operating modes and its inputs activity characteristics. The proposed online monitoring mechanism drastically reduces the communication-derived latency between the monitor and the DVFS. Experimental results show a mean absolute percentage error below 1.5% for the estimated power consumption.

Keywords: Data generation and acquisition · power measurement · FPGA · power estimation and modeling · machine learning · artificial neural networks · online power monitor · DVFS

1 Introduction

Today, power consumption is one of the first and main constraints when designing digital hardware systems such as FPGAs or ASICs. In general, power estimation is obtained by evaluating the signals' activity of a given circuit in a scenario of execution that runs within a certain time frame. In parallel, online power monitoring is becoming indispensable for controlling the power consumption of large digital circuit systems at runtime.

Runtime adaptive systems determine dynamically and autonomously the optimal operating points for power consumption. Such systems take

application-specific requirements into account and specifically adapt the workload. Traditionally, current sensors (either built-in or external) have been used to periodically, under software control, collect power information to be transferred to variable voltage regulators and/or configurable Phase-Locked Loops (PLLs) or simply Voltage-Controlled Oscillators (VCOs). These latter, under the DVFS mechanism, regularly tune the core supply voltage and/or the input clock frequency of the device under optimization. Appropriate design of DVFS-based applications can result in up to 65% improvement in energy consumption [3]. The main issue to tackle remains in the communication overhead that can have undesirable effects on a running application on FPGAs; notably the latency between the sensors and the DVFS, and also between the DVFS from one side and the variable voltage regulators and/or the tunable frequency devices from the other.

In this work, we apply a previously proposed machine-learning automated data training construction methodology leading to high-level FPGA IP power estimation [1] in order to create power models supporting the online power monitoring and subsequent management. Here, we should mention that the power monitor/estimator in context is implemented per FPGA IP thus providing the ability to monitor individual IPs' power consumption. Most importantly, the proposed scenario eliminates the latency effect between the power estimator and the DVFS. In addition, it accelerates the response between the DVFS from one side and the variable voltage regulator and/or tunable frequency devices from the other side when applying the minimal latency path.

This paper is organized as follows: Sect. 2 elaborates on recent related works. Section 3 presents our proposed power consumption modeling and estimation methodology as well as the online power monitoring approach. Section 4 reveals the experimental results where a black-box IP with different scenarios and operation modes was put under test. Subsequently, the model's assessment is highlighted. Finally, we conclude in Sect. 5.

2 Related Work

With the high demand on fast and complex FPGA-based systems and their dominance in the digital world (embedded systems, communication linecards, data centers, cloud computing, etc.), online power consumption monitoring is no longer a nice-to-have feature but an indispensable requirement.

Some works aim at supporting emerging power management techniques, for example fine grained DVFS. In [7], a dedicated hardware circuit is proposed to obtain internal signal activities during runtime while predicting power consumption on-the-fly. The authors claim that dynamic power may be estimated within an error margin of 1.90% compared with commercial gate-level power estimation tool. The major limitation though is that the model is trained on simulated data from the Vivado power analyzer, which does not take into consideration the real conditions of execution.

At RTL, most recent works that study high-level power modeling of FPGAs usually rely on linear regression methods. For example, in [10] or [4] the approach

consists in monitoring influential signals within specific modules. Power consumption is then measured and a linear power model is built and updated online. The main drawbacks of these previous works is the lack of accuracy of the proposed models which rely on simple linear mathematical models.

In this work, compared to most related methods and in order to build robust training data sets for the learning-based estimation, we rely on real power values obtained from a physical acquisition system, under software control. Subsequently, we estimate, in real-time, FPGA IP power consumption, based on the coupling of the IP's Most Significant Modes (MSM) of operation and the activity of its data path inputs. Subsequently, the obtained results show wider adaptivity and higher accuracy.

3 Methodology

Online power monitoring of a digital circuit is the process of periodically collecting its energy usage for subsequent optimization using well-known mechanisms such as DVFS. The proposed online power monitoring methodology aims at estimating a given FPGA IP power consumption in-situ and in real-time. It is based on machine learning and precisely on supervised Artificial Neural Networks (ANN) [5].

Traditionally, power consumption monitors (also known as current sensors), whether built-in or external, measure the total consumed energy of the FPGA's core and cannot differentiate between various co-existing IPs' consumption. In general, these sensors operate over the Power Management Bus (PMBus). The PMBus is a variant of the System Management Bus (SMBus) which is targeted at digital management of power supplies. It is a relatively low speed, two-wire serial communication protocol, based on Inter-Integrated Circuit (I²C) [11]. This also applies to the voltage/frequency controllers (variable/configurable regulators/PLLs/VCOs), also operating over the same PMBus. That said, and as an undesirable effect, this scenario injects delays in the power monitoring and subsequent management system's response. For instance, a PMBus clocked at 400 KHz (fast mode) produces, at best, a latency of 300 μ s per one iteration, assuming no (embedded) software overhead is present [6].

Figure 1 provides a high-level overview of the proposed power monitoring and management mechanism. The power monitor in context, considered as our main concern, is divided into two blocks. The first block extracts: 1) the input activity of the IP's data path in terms of the switching rate and the percentage of logic "1" occurrences and 2) the most energy-significant modes of operation (MSM) that can be derived from the control signals of the IP's state machine or even, in some cases, can be explicitly accessible. The second block is the power estimator itself having the extracted input activity parameters along with the most significant modes of operation as inputs. This latter consists of the implementation of an artificial neural network inside the FPGA. Both the implementation details and the neural network architecture are discussed in later subsections throughout the paper (Subsects. 3.2 and 3.3).

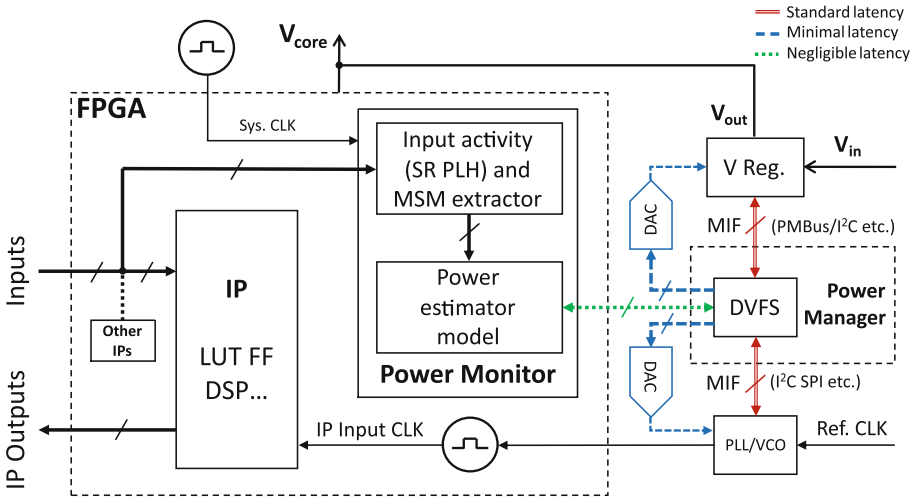


Fig. 1. Power monitoring and management mechanism along with latency options

Figure 1 also reveals the existing latencies at various locations of the proposed model. The latency between the power estimator and the DVFS mechanism depends on the number of cycles of the power estimator itself from the moment it captures the inputs until it generates an output. This work aims at keeping this specific latency at the very minimum. The latency between the DVFS and the tunable devices (voltage and/or frequency) is highly dependent on the Management Interface (MIF). Standard communication latency is observed when dealing with PMBus/I²C protocols, and minimal latency is produced when dealing with fast Digital to Analog Converters (DACs), as data are directly driven from discrete fast outputs. Note that this specific type of latency is out of the scope of this work. As for the DVFS system itself, it may be decomposed into a set of comparators acting on pre-defined power consumption threshold values and dynamically tuning the corresponding parameters within pre-defined operating limits. That said, the DVFS' operation does not add significant latencies to the power monitoring and management loop (power estimator → DVFS → tunable devices and again.. ↔). The proposed methodology consists of three sequential steps:

1. IP characterization using a well-defined high fidelity platform.
2. Build of the training data sets using the collected power information and subsequently build of the power model.
3. Implementation of the neural network inside the same FPGA target.

3.1 IP Characterization System

FPGA IP characterization reveals the distinctive nature of a given target digital circuit and highlights its special features such as resources and most importantly

its power consumption information. Here we should point out that, as shown in Fig. 2 (right-hand side), an FPGA IP can be decomposed into a Finite State Machine (FSM) fed by a certain number of Control Signals (CS), and a Data Path (DP) fed by the actual inputs toggling at specific rates.

The IP's power consumption, during well-defined Operating Modes (OMs), is highly affected by its input switching activity mainly the Switching Rate (SR) and the Percentage Level High (PLH). The SR represents the ratio of transitions in a given bit sequence, whereas the PLH is the percentage of logic "1" bits in the same sequence. Operating modes are specific functional behaviors of a given IP (for ex: idle mode, half-duplex mode, full-duplex mode, loopback mode, burst mode, etc.), out of which we select the most power-hungry subset denoted as MSM. These operating modes are in a close correlation with the control signals feeding the IP's FSM, or in some other cases, they are explicitly exposed. In Fig. 2 (right-hand side) the relationship between the control signals and the operation modes is represented by a decoder (X/Y). In order to collect precise power information, a reliable measurement platform is required. For that purpose, we have proposed an FPGA-based Automated and Centralized Data Generation and hybrid Acquisition System (ACDGAS) combining the features of three instruments: a sampling oscilloscope with analog inputs, a logic analyzer and a bit-pattern generator [2]. As shown in Fig. 2, High Speed Digital I/Os (HSDIO) are feeding the DUT-FPGA (configured for 1 IP only) as 16-bit input signals and the power consumption measurement is sampled via a high speed parallel differential Analog to Digital Converter (ADC) through a precision shunt resistor R_S on the FPGA core voltage (1V).

The platform's main role is to apply stimuli signals at the IP inputs and synchronously collect aligned power consumption samples under software control. The proposed layered software architecture delivers a fully automated process for IP characterization. It provides a solid and synchronized interaction between the various system modules such as: graphical user interface, stimuli construction, MSM coupling, bit sequence generation, measurement platform interface, training data construction, power modeling and evaluation.

3.2 Proposed Model

Power or energy system modeling is the process of building abstract models to perform analysis and subsequently estimate power consumption of digital circuits according to specific criteria. For many machine learning techniques, especially the ones related to supervised methods, the construction of the training data highly affects the quality and accuracy of the derived model [8]. In our case, the collected data sets are derived from two different sources: the stimuli generation algorithms and a hardware data acquisition system providing real power consumption values collected after applying the generated stimuli on a given IP.

Regarding the parametric stimuli generation as shown in Fig. 3, two sets of PLH and SR of H and S values are respectively used to generate $M = H \times S$ distinct combinations of [PLH, SR] pairs. The aforementioned combinations are

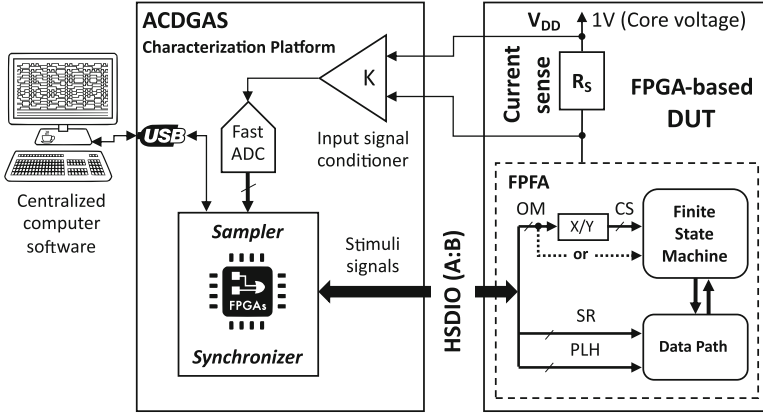


Fig. 2. Measurement and characterization system hardware setup along with the IP under test

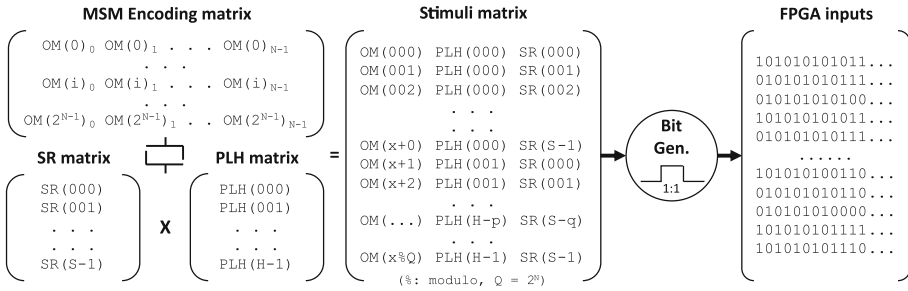


Fig. 3. Stimuli generation matrices along with the actual FPGA input bits

coupled with N encoding bits for the MSM thus leading to 2^N modes of operation. Each permutation of the MSM bits is appended to I pairs of [PLH, SR], subsequently generating one stimuli matrix. Each pair of [PLH, SR] generates 1 fixed-length sequence of bits that will be eventually feeding a given IP's data path input. The bit sequence generator in context ensures the [PLH, SR] pairs' compatibility and fulfills the stimuli signal's requirements while achieving a high degree of entropy in the generated bits (adequate SR distribution and minimal bit-clustering). As a direct application, specific values are selected in order to implement the proposed model: $H = 200$, $S = 300$, $M = 60,000$, $N = 4$ bits, $I = 12$ bits, $D = M \div I = 5,000$ and IP DP inputs width = 1 Kbit. The resulting training data set consists of 5,000 entries of 24 (12×2) inputs representing pairs of SR and PLH coupled with 4 binary inputs representing the encoding of the MSM, in addition to the (10-bit) raw output average power.

The proposed neural network architecture is compiled under the TensorFlow/Keras Python library. As shown in Fig. 4 (right-hand side), it consists of the input layer fed by 4 binary inputs (OM_i) resulting in $2^4 = 16$ modes

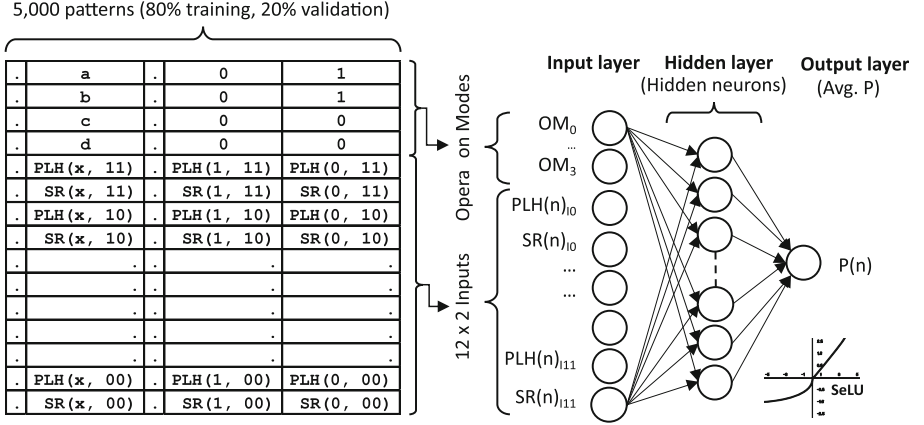


Fig. 4. Neural network architecture showing training data, layers, neurons and activation function (SeLU)

and 12 pairs of SR and PLH, one hidden layer and one output layer representing the estimated (average) output power. The number of hidden neurons has been optimized following the FPGA hardware implementation usage presented and discussed in Subsect. 3.3 and Table 1, providing a trade off between training/prediction speed and accuracy. The Scaled Exponential Linear Unit (SeLU) is chosen as the activation function due to its self normalizing nature and its high learning robustness [9]. As shown in Fig. 4 (left-hand side), the training set size represents 80% of the total data set corresponding to 4,000 patterns, out of which, 3200 (80%) are used for training and 800 (20%) for validation. The remaining samples are reserved for evaluating the proposed neural network model, and represent 20% (1,000 samples) of all available patterns. The network optimizer is selected to be *Adam* that is a replacement optimization algorithm for stochastic gradient descent for training deep learning models [13]. The adopted training loss metric is the Mean Squared Error (MSE) while the evaluation metrics are the Mean Absolute Error (MAE) and the Mean Absolute Percentage Error (MAPE). The number of training epochs is optimized to be 25 iterations.

The automated power modeling process and data flow are illustrated in Fig. 5. The procedure encapsulates the following sequential steps:

1. Generation of MSM encoding bits, [PLH, SR] tables and combinations, and resulting stimuli matrices.
2. Generation of stimuli files, creation of subsequent ACDGAS batch-job and execution.
3. Collection of power consumption samples (in hardware) and construction of training data sets.
4. Data normalization, neural network compilation, training and evaluation.

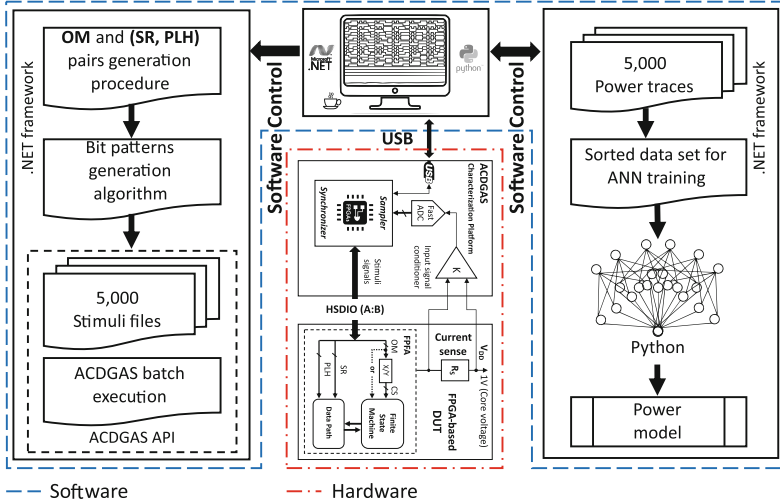


Fig. 5. Automated work and data flow showing data generation and acquisition on the left-hand side, acquisition platform in the middle and, ANN training and power modeling on the right-hand side

3.3 Hardware Implementation

Implementing an artificial neural network (ANN) inside the FPGA can be done either by software via an embedded processor or by hardware using a dedicated digital circuit implementation [12]. In order to accelerate the proposed ANN and subsequently eliminate its prediction latency effect, the hardware implementation is a must. This latter can be pipelined while processing in parallel and in real-time. However, a trade off between FPGA logic usage and prediction accuracy/speed is inevitable. Table 1 shows the neural network usage and latency (in cycles) when implemented using 4, 8 and 16 hidden neurons respectively. To provide additional implementation flexibility, at the input layer, data are scanned in batches of either 512 or 1024 bits per iteration in order to extract PLH and SR and subsequently estimate power consumption. Processing at 100 MHz, the recorded latencies show that the neural network is performing in real-time except for the 16 (*) hidden neurons version when operating in 512 bits per iteration. In this case the initial interval cycles surpass the number of bits per iteration. The recorded power estimator latencies ($< 10 \mu\text{s}$ for 512 bits and $< 20 \mu\text{s}$ for 1024 bits), compared to most standard communication-based overhead, showed an improvement of above 90%. The target FPGA used is the AMD/Xilinx Artix-7 (xc7a35tftg256-2) under the High-Level Synthesis (HLS) tool provided by Vitis HLS 2022.2 using an off-the-shelf neural network IP.

Table 1. ANN hardware implementation: usage and latency

Bits per iteration	Hidden neurons	Initial interval (cy)	Latency (cy)	DSP	FF	LUT
512	4	512	926	12	6,104	11,936
	8	512	978	12	6,389	11,918
	16*	850*	1,362	12	6,396	11,926
1024	4	1,024	1,438	12	6,105	11,938
	8	1,024	1,559	12	6,218	11,891
	16	1,024	2,023	12	6,226	11,899

4 Experimental Results

In order to evaluate the proposed power consumption estimation model, it is essential to conduct experimental tests. We have applied our methodology on FPGA circuits and collected power information in order to build the power model and record the obtained results. An off-the-shelf AMD/Xilinx Artix-7 (xc7a35f5tfg256-2) FPGA running at 100 MHz was used in this context. Here we should mention that the total power consumption of an IP is derived from its dynamic and static power added together. The dynamic one occurs when the IP signals are toggling while the static one is always present regardless of its activity. In the following test cases the dynamic power is dominant.

4.1 Test Cases

Since the proposed methodology is not aware of the FPGA IP under monitoring (functionality, components, interconnections, etc.) and in order to generalize our real-time power estimator, we have selected a black-box FPGA IP to put under test. A black-box IP is a digital circuit whose functionality and internal connections are masked. As stated earlier, the online power estimator relies only on the extracted switching activity of the IP's data path inputs while being coupled with a certain number of operation modes. These latter can be either extracted from the control signals of the same IP's state machine or simply provided by the designer. The black-box IP in context has 12 inputs and 16 modes of operation encoded over 4 bits. It has been tested under two different power estimation models, each generated using a specific neural network implementation: one with 4 hidden neurons and one with 8. The 16-neuron alternative has been discarded due its relatively high latency.

4.2 Model Assessment

Figure 6 represents the training loss and the MAPE of both data training and model validation for the different power estimators. The 4 hidden neurons version is shown in (a) whereas the 8 hidden neurons model is shown in (b). Both the MSE and MAPE are displayed on the vertical axis of each graph pair respectively. The number of training iterations denoted as Epochs is represented on the

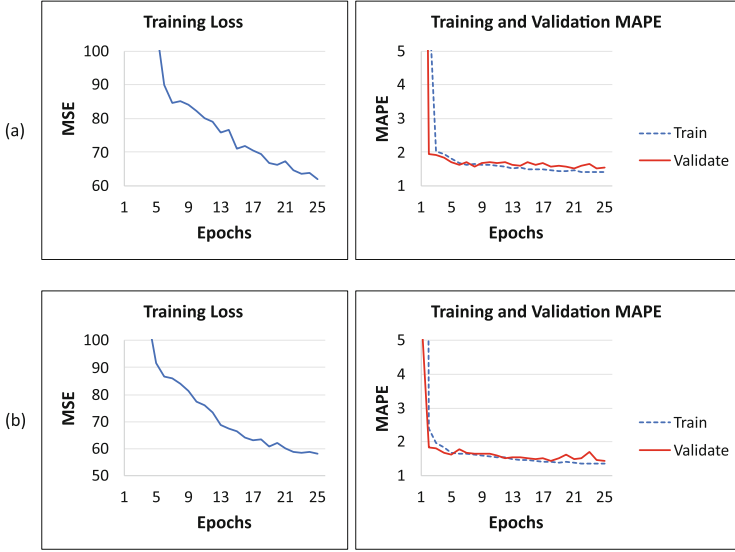


Fig. 6. Learning curves for each power model showing training loss and MAPE (training and validation) v/s number of epochs

Table 2. Resources and performance results

	Black-box IP	
IP utilization	LUT	14296
	FF	3761
	DSP	50
Min. avg. power (mW)	25.00	
Max. avg. power (mW)	94.50	
Estimation MAE	4 neurons	6.67
	8 neurons	6.00
Estimation MAPE	4 neurons	1.55%
	8 neurons	1.30%

horizontal axis. As the number of epochs increases, both the prediction loss and the mean absolute percentage error decrease drastically, proving the efficiency of the proposed architecture. Beyond 25 epochs, the training loss (MSE) and the validation (MAPE) values hit a steady state.

Table 2 combines the IP usage (LUTs, FFs and DSPs), the estimated average power consumption range (minimum and maximum) and the power model evaluation metrics (MAE and MAPE) for 4 and 8 hidden neurons respectively. The MAE values designate the absolute error in raw power levels ranging between 6 and 6.67, out of 1024 total levels (10-bit resolution). The MAPE is the estimation

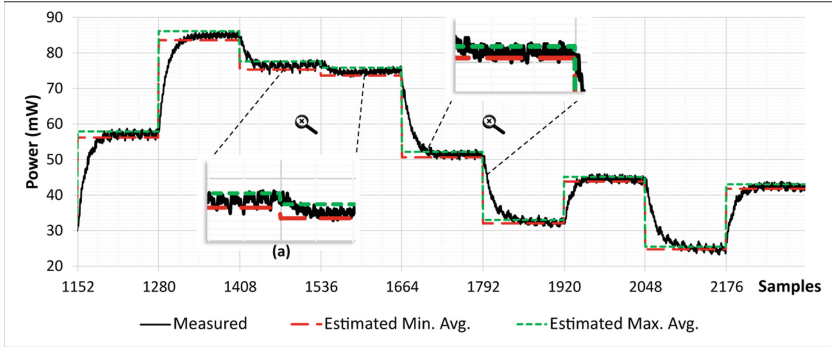


Fig. 7. Per-mode measured v/s estimated power consumption

absolute percentage error being less than 1.5% for both cases. With the 8 hidden neurons model an estimation improvement of 16% is recorder over the 4-neuron alternative model. As an observation, the proposed power estimation methodology is highly efficient on LUT- and DSP-based IP circuits. The experimental results show fast and very accurate estimation values when applied to black-box IPs. This proves the robustness as well as the coherence of the presented neural network’s architecture, and subsequently, the high adaptivity of the presented online power monitoring technique.

Figure 7 shows multiple unsorted operation modes for the black-box IP along with their respective measured and estimated power consumption levels in mW. Each permutation of the OM bits, coupled with the input activity of the data path, yields to a functioning mode with a specific power level. For instance, a specific value of the OM bits leads to an operation mode (spread over 128 samples) with variable power levels depending on the data path input activity (PLH and SR). Subsequently, we may notice different average power values for the same operation mode. The estimated power is represented by the maximum and minimum average levels following the \pm prediction percentage error. In Fig. 7(a), two neighboring operating modes are detected with very close power levels, yet the estimator was able to differentiate and to properly estimate the corresponding power consumption.

5 Conclusion

Power consumption optimization is indispensable for modern digital hardware especially with high speed and increasing complexity. Online power monitoring and subsequent management is becoming a hot topic since it provides on-the-fly energy optimization. In this work, we leverage the machine learning techniques to establish an efficient neural-network-based online power monitoring and estimation approach for IPs in FPGAs. We estimate the power consumption in-situ and in real-time by just providing the most energy-significant modes of operation

coupled with the input activity defined by the switching rate and the percentage of logic high bits of individual black-box FPGA IPs. The experimental results confirm the model's accuracy with a mean absolute percentage error below 1.5% and a minimum of 90% latency improvement between the power monitor and the DVFS controller. Additional optimizations on the ANN implementation inside the FPGA could also lead to a smaller hardware footprint and thus higher efficiency.

References

1. Richa, M., Prévotet, J.-C., Dardaillon, M., Mroué, M., Samhat, A.E.: High-level early power estimation of FPGA IP based on machine learning. In: 29th IEEE International Conference on Electronics, Circuits, and Systems (ICECS), pp. 1–4 (2022). <https://doi.org/10.1109/ICECS202256217.2022.9970952>
2. Richa, M., Prévotet, J.-C., Dardaillon, M., Mroué, M., Samhat, A.E.: An automated and centralized data generation and acquisition system. In: 28th IEEE International Conference on Electronics, Circuits, and Systems (ICECS), pp. 1–4 (2021). <https://doi.org/10.1109/ICECS53924.2021.9665490>. ISBN 9781728182810
3. Akgun, G., Ali, M., Gohringer, D.: Power-aware computing systems on FPGAs: a survey. In: 31st International Conference on Field-Programmable Logic and Applications (FPL), pp. 45–51 (2021). <https://doi.org/10.1109/FPL53798.2021.00016>
4. Davis, J.J., Hung, E., Levine, J.M., Stott, E.A., Cheung, P.Y., Constantinides, G.A.: KAPow: high-accuracy, low-overhead online per-module power estimation for FPGA designs. *ACM Trans. Reconfigurable Technol. Syst.* (2018). <https://doi.org/10.1145/3129789>
5. Gallo, C.: Artificial neural networks. In: *Convolutional Neural Networks in Visual Computing*, pp. 1–426 (2011). <https://doi.org/10.4324/9781315154282-3>
6. Hesse, K.: Using PMBus for improved system-level power management. In: *Texas Instruments Seminars*, p. 19 (2008). www.ti.com/download/trng/docs/seminar/Topic_6_Hesse.pdf
7. Lin, Z., Sinha, S., Zhang, W.: An ensemble learning approach for in-situ monitoring of FPGA dynamic power. *IEEE Trans. Comput. Aided Des. Integr. Circ. Syst.* **38**(9), 1661–1674 (2019). <https://doi.org/10.1109/TCAD.2018.2859248>
8. Mange, J.: Effect of training data order for machine learning. In: 2019 International Conference on Computational Science and Computational Intelligence (CSCI), pp. 406–407 (2019). <https://doi.org/10.1109/CSCI49370.2019.00078>
9. Marcu, D.C., Grava, C.: The impact of activation functions on training and performance of a deep neural network. In: 2021 16th International Conference on Engineering of Modern Electric Systems (EMES), pp. 1–4 (2021). <https://doi.org/10.1109/EMES52337.2021.9484108>
10. Najem, M., Benoit, P., Bruguier, F., Sassatelli, G., Torres, L.: Method for dynamic power monitoring on FPGAs. In: *Conference Digest - 24th International Conference on Field Programmable Logic and Applications, FPL 2014* (2014). <https://doi.org/10.1109/FPL.2014.6927457>
11. White, R.V.: PMBUs: a decade of growth: an open-standards success. *IEEE Power Electron. Mag.* **1**(3), 33–39 (2014). <https://doi.org/10.1109/MPPEL.2014.2330492>

12. Yi, Q.: FPGA implementation of neural network accelerator. In: 2018 2nd IEEE Advanced Information Management, Communicates, Electronic and Automation Control Conference (IMCEC), pp. 1903–1906 (2018). <https://doi.org/10.1109/IMCEC.2018.8469659>
13. Zhang, Z.: Improved Adam optimizer for deep neural networks. In: 2018 IEEE/ACM 26th International Symposium on Quality of Service (IWQoS), pp. 1–2 (2018). <https://doi.org/10.1109/IWQoS.2018.8624183>

Author Index

B

Beck, Antonio Carlos Schneider 95

C

Cebrián, Pedro L. 15, 28

Chavarrías, Miguel 15, 28

D

Dardaillon, Mickaël 107

de Oliveira Rubiano, Daniel 95

Desnos, Karol 3

Dias, Tiago 40

G

Gageot, Dylan 3

Grabowski, Mariusz 70

J

Juarez, Eduardo 15, 28

K

Koch, Andreas 82

Korol, Guilherme 95

Krichene, Hana 55

Kryjak, Tomasz 70

L

Lagares, Alfonso 15, 28

M

Malcata, Tomás 40

Martinez de Ternero, Alejandro 15

Martín-Pérez, Alberto 15, 28

Meisel, Florian 82

Mouhagir, Ayoub 55

Mroué, Mohamad 107

N

Nezan, Jean-François 3

P

Prasad, Rohit 55

Prévotet, Jean-Christophe 107

R

Renaud, Ophélie 3

Richa, Majdi 107

Roma, Nuno 40

Rosa, Gonzalo 15, 28

S

Samhat, Abed Ellatif 107

Sancho, Jaime 15, 28

Sanz, César 15, 28

Sebastião, Nuno 40

Spang, Christoph 82

Sutradhar, Pallab 15, 28

T

Ternero, Alejandro Martinez de 28

Tran, Dat 82

V

Vazquez, Guillermo 15, 28

Villa, Manuel 15, 28

Volz, David 82