Application Specific Instruction-set Processors for Massive MIMO Systems

Attari, Mohammad

2024

Document Version:
Publisher's PDF, also known as Version of record

Application Specific Instruction-set Processors for Massive MIMO Systems

Doctoral Thesis

Mohammad Attari

LUND
UNIVERSITY

Department of Electrical and
Information Technology
Lund, October 2024

Academic thesis for the degree of Doctor of Philosophy (PhD), which, by due permission of the Faculty of Engineering at Lund University, will be publicly defended on Monday, 11 Nov, 2024, at 9:15 a.m. in lecture hall E:1406, Department of Electrical and Information Technology, Ole Römers Väg 3, 223 63 Lund, Sweden. The thesis will be defended in English.

The Faculty Opponent will be professor Andreas Peter Burg, Ecole Polytechnique Federale de Lausanne (EPFL), Switzerland.

Organisation:
LUND UNIVERSITY
Department of Electrical and Information Technology
Ole Römers Väg 3
223 63 Lund
Sweden

Author:
Mohammad Attari

Document Type:
DOCTORAL THESIS

Date of Issue:
October 2024

Sponsoring Organization(s):
Ericsson AB
European Union REINDEER No. 101013425
BEYOND5 No. 876124

Title:
Application Specific Instruction-set Processors for Massive MIMO Systems

Abstract:

This is an undeniable fact now that wireless systems pervade all aspects of our lives. These systems are evolving at a rapid clip, connecting more people and devices every single day that goes by. This growth is further fueled by the users' insatiable appetite for more traffic, be it for online gaming, watching high-fidelity video, downloading huge files, live-streaming and many more uses. With the advent of internet of things (IoT), which brings a countless number devices and sensors into the picture, this growth turns into an unstoppable force.

Catering to the connectivity and data rate demands that these applications and devices place on the wireless communications infrastructure is not a trivial issue. As the old 4G systems are approaching, or rather have already surpassed, their limits, the new kids on the block are 5G and what comes beyond. These systems are developed specifically to bump up the data rates, provide better coverage, and increase the overall energy and spectral efficiencies. In order to facilitate this, a number of key technologies have proven themselves instrumental. One such technology is the massive multiple-input multiple-output (MIMO), which scales up the number of antennas available in the base station (BS) to the hundreds, in order to add *space* as yet another degree of freedom to the system, creating the holy trinity of time-frequency-space. This is crucial, considering the fact that frequency resources are limited, very expensive, and already overcrowded. This idea can be, and is being, pushed even further by employing thousands of antennas in systems such as large intelligent surfaces (LISs).

But it is not all moonlight and roses, as one might think. Incorporating these many antennas in the system puts a huge burden on data processing and data marshaling subsystems. A centralized approach does not carry the day here, and distributing the processing is not a piece of cake either. That is what this thesis concerns itself with, i.e., how to develop processors that are up to par with the requirements of above-mentioned systems in terms of performance and energy efficiency, yet are malleable enough to adapt to the vagaries of technological evolution. To this end, processor designs have been proposed here that utilize application-specific instruction set processors (ASIPs) as the firm ground to build the system upon, which are wedded to customized accelerators where more specialized units are deemed more appropriate to tackle the case at hand.

Keywords:
Massive MIMO, OFDM, Digital Baseband Processing, 5G and Beyond, VLSI Architecture, Computer Architecture, ASIP, Accelerator, Systolic Array, Matrix Decomposition, SIMD

Classification System and/or Index Terms (if any)
–

Language:
English

Supplementary Bibliographical Information:
–

ISBN (printed):
978-91-8104-247-4

Key title and ISSN:
Series of Licentiate and Doctoral Theses; 1654-790X, No. 177

ISBN (digital):
978-91-8104-248-1

Recipient's Notes:

Number of Pages:
122

Price:

Security Classification:
Unclassified

Signature: 

Date: 11 October 2024

Application Specific Instruction-set Processors for Massive MIMO Systems

Doctoral Thesis

Mohammad Attari

LUND
UNIVERSITY

Department of Electrical and
Information Technology
Lund, October 2024

Mohammad Attari
Department of Electrical and Information Technology
Lund University
Ole Römers Väg 3, 223 63 Lund, Sweden

Series of Licentiate and Doctoral Theses
ISSN 1654-790X; No. 177
ISBN 978-91-8104-247-4 (printed)
ISBN 978-91-8104-248-1 (digital)

This thesis is typeset using LaTeX 2ε with the body text in Palatino and Goudy Initials, headings in Helvetica.

Frontispiece: Hand-drawn interpretation of communications and processing-related concepts.

Printed by Tryckeriet i E-huset, Lund University, Lund, Sweden.

"The lyf so short, the craft so long to lerne,
Th'assay so hard, so sharp the conqueringe"

Geoffrey Chaucer, Parlement of Foules

Abstract

It is an undeniable fact now that wireless systems pervade all aspects of our lives. These systems are evolving at a rapid clip, connecting more people and devices every single day that goes by. This growth is further fueled by the users' insatiable appetite for more traffic, be it for online gaming, watching high-fidelity video, downloading huge files, live-streaming and many more uses. With the advent of Internet of Things (IoT), which brings a countless number devices and sensors into the picture, this growth turns into an unstoppable force.

Catering to the connectivity and data rate demands that these applications and devices place on the wireless communications infrastructure is not a trivial issue. As the old 4G systems are approaching, or rather have already surpassed, their limits, the new kids on the block are 5G and what comes beyond. These systems are developed specifically to bump up the data rates, provide better coverage, and increase the overall energy and spectral efficiencies. In order to facilitate this, a number of key technologies have proven themselves instrumental. One such technology is the massive Multiple-Input Multiple-Output (MIMO), which scales up the number of antennas available in the Base Station (BS) to the hundreds, in order to add *space* as yet another degree of freedom to the system, creating the holy trinity of time-frequency-space[1]. This is crucial, considering the fact that frequency resources are limited, very expensive, and already overcrowded. This idea can be, and

[1]OK, it's not really holy, but who's counting, so I claim it is.

is being, pushed even further by employing thousands of antennas in systems such as Large Intelligent Surfaces (LISs).

But it is not all moonlight and roses, as one might think. Incorporating these many antennas in the system puts a huge burden on data processing and data marshaling subsystems. A centralized approach does not carry the day here, and distributing the processing is not a piece of cake either. That is what this thesis concerns itself with, i.e., how to develop processors that are up to par with the requirements of above-mentioned systems in terms of performance and energy efficiency, yet are malleable enough to adapt to the vagaries of technological evolution. To this end, processor designs have been proposed here that utilize Application-Specific Instruction-set Processors (ASIPs) as the firm ground to build the system upon, which are wedded to customized accelerators where more specialized units are deemed more appropriate to tackle the case at hand.

Popular Science Summary

It would not surprise you these days if you pass a bus stop and see 9 people out of 10 having their heads buried deep into their cellphones. What on Thor's green earth could they be doing? Well, of course, they are communicating information wirelessly in one form or another. It could be that they're blundering their pieces left and right in a chess game online, or chatting with someone, or taking their daily dose of funny memes on social media, or reading the news, or live-streaming, or a gazillion other things that I can not possibly list here.

One of the myriad of factors that separates us humans from other living beings is the unique way in which we convey information, either through language or by non-linguistic means. This desire to carry a message across has existed since the dawn of our existence and, throughout our rather short evolutionary history, we have always dreamed of communicating over long distances. The smoke signals are one of the earliest forms of long-distance communications that we came up with, yet it was slow and unreliable. Much later, and as fate would have it, it was via wired means that long-distance communication became a reality. Although it didn't take long for the information we were sending to be carried in the air through the magic of radio waves, ushering in the wireless communications age.

Nowadays you can not imagine your life without the wireless technology. It has permeated so thoroughly through the fabric of our society that people take it for granted and don't even think about how marvelous it is. Now talking to your loved ones across the globe is just a few taps away on your cell-phone. In 2020 the world was hit by the COVID pandemic, but the amazing advances

in communications made it possible for people to stay safe inside their homes and even work form there. It is not just personal applications that is driving the wireless communications. The fast-paced nature of business today is a huge contributing factor.

The ease with which we can communicate means there is a huge amount of information that needs to scuttle back and forth. To satiate this huge traffic demand, engineers and researchers have been hard at work to come up with mechanisms to keep up. This includes the use of fancy-sounding techniques such as massive MIMO, which incorporates hundreds of antennas to make better use of the limited resources. Large intelligent surfaces are yet another technique that try to scale up the number of antennas even further. Having said all this, the challenges of using these techniques are not easy to overcome. The processing of data that go through the network poses stiff hurdles to overcome.

The main processors that run under the hood in your laptop or cell-phone are so-called general-purpose processors. These processors are good at doing a wide variety of things, and are capable of running a whole host of applications, but are not particularly adept at executing very specific tasks efficiently. That is where this thesis comes into the picture. In this work, I have investigated the usage of custom-made processors specifically geared for this kind of custom digital signal processing, while retaining, to some extent, the flexibility of the afore-mentioned general-purpose processors. The kind of processing that is needed mostly involves fiddling with numbers arranged in neat packages known as matrices and vectors. We need to crunch these numbers, using algebraic algorithms, fast and efficiently, therefore customized designs are a must. But in a world that change is the only constant, it is not only desirable, but rather mandatory to keep the system programmable and flexible to account for the vagaries of the algorithms and technology. As a result, application-specific instruction set processors, or so-called ASIPs, were chosen and developed. Additionally, to further bolster their performance, non-programmable and highly-customized-yet-configurable accelerators were designed to aid the ASIPs in taking on compute-heavy processing tasks, and in unison act as a conduit to our digital wireless future

Acknowledgments

THANKS to the teachings passed down to us from the wise sages of yore, it is nowadays common knowledge that completing a PhD is not a solely individual endeavor. Well, for the most part it is, but you get the picture. There are many people along the way who make it easier[2] for you to traverse this treacherous path to Mount PhD, and they deserve their fair share of fame as yours truly. To do these people justice even multiple tomes would not suffice, alas, I'm pressed for space here, and I hope the PhD gods will take mercy on my poor soul for brazen brevity and negligence.

I admit it, I'm not very good at verbalizing thanks, yet what follows is my meager attempt at it. First and foremost, I want to thank my main supervisor, **Liang**, for his patience and guidance. I am the one who tread the path but he was the one pointing the way. I know I can be pigheaded sometimes[3], so Liang is probably a saint for overlooking this foible. I am grateful for your efforts to try to take me out of my hermit lifestyle by taking me to the chess club, and for inviting me over to your house and introducing me to your two little princesses, **Linea** and **Linda**. I would also like to express my thanks to my co-supervisor, **Ove**, for technical discussions and for giving constructive feedback on my work.

Now, let's march on to my colleagues and friends in Lund University and beyond. Here comes the cavalry, in a sort of semi-haphazard order:

Mr. B (a.k.a. **Baktash Behmanesh**) deserves super special thanks for providing me with comic relief during my PhD, as well as for Being[4] a fellow gamer. You taught me how to show those metal tracks a lesson during DRC

[2] Or harder, depending on the deviousness level.

[3] Confucius he say: "all the frigging time".

[4] Yes, everything Mr. B-related with capital B.

cleaning, which I knew zilch aBout Beforehand, and you also turned me into a soldering master. The chip I taped out would not have Been possiBle without your assistance. To Mr. B's much, *much* Better half, **Mrs. B** (**Behshid Khodaei**), I say thanks for Being supportive and also for putting up with Mr. B. I salute **Iman Pasha** (also goes by **Iman Ghotbi**) for always regaling us with his impressive analytical ability, and also for bringing us cookies and other things to stuff our faces with. I appreciate the cool and collected **Hamid**, for bringing tranquility and serenity to the highly volatile field of electronics. May you ever remain as cool as a cucumber. Many thanks go to **Major** (the honorable **Masoud Nouripayam**) for being passionate and showing strength and resilience in the face of hardship, and also for giving me beautiful plants to liven up my austere-looking office.

I send my gratitude to **Luki Boy** (others might know him as **Lucas Ferreira**) for spicing up the work environment and talking my (and everyone else's) head off during office hours. I value the time we have spent together and look forward for its continual into the posterity. I'm much obliged to **Arturo** (the Spaniard) for never taking the bait on my taunts of "you wanna take it outside?". Everyone knows how that might have turned out[5]. I'm grateful to **Siyu** for the help I got during the torturous last week before my tapeout, showing me how to run the dreaded LVS. There would be no working chip without it. I'm thankful to **Jesús** for his collaborative spirit and his patience in working with me. I am indebted to **Dr. Steffen** (the one and only **Steffen Malkowsky**) for being there at the start of my PhD (and even before that, in my PhD interview) and for showing me the ropes with ASIP Designer. Now, stop watching documentaries! Many thanks to **Dumitra** for being super cool and inviting me to go climbing with her. Thanks **Krippe** (recognized in some circles with the moniker **Kristoffer Westring**) for nice chats and encouraging me to improve my Swedish. Please stop bragging about your chip; mine is better. I thank **Ilayda** for the delicious baklava she brought me and for inviting me to the barbecue she organized. To **Mahdi Rezayati Charan**, I say thanks for letting me, unwittingly, dribble elegantly past him in a soccer game, only to be tackled mightily half a heartbeat later. Thanks for the fun times and for inviting me over to your place. **Sirvan** is praiseworthy for setting me up in the lab, fixing my PCB, and meticulously bonding my chip. I raise my hat to **Mazi** and **Aida** for their help and kindness throughout the years. I greet my friend **Harsh** (**Harshavardhan Kittur**) with cheers for encouraging me to soldier on in the good old days, and accompanying me to super-duper rigorous swimming competitions in the gooder[6] new days.

[5] In my favor. All hail!

[6] Better.

I send my small tokens of appreciation to the following people: **Vincent** for being kind and letting us use the 1406 room for the defense, **Juan** (of house **Vidal Alegría**) for chess discussions, **Juan** (of house **Sanchez**) for patiently answering my badly formulated 5G questions, **Rikard** for being my BFF[7], **Therese** for being kind and inviting me to her PhD party, **Sijia** for sharing her work on GitHub, **Lina** for being a good office-mate, **William** for lighthearted banter, **Sahar** for tolerating my OCD and keeping it cool with my stupid jokes, **Mojtaba** for supportive and reassuring words early on in my PhD, **Ashkan** for not tackling me in soccer unlike some people, **Peng** and **Feifei** for treating me to dinner, **Wei** for inviting me over to his place, and **Sidra** for the nice conversations.

I owe thanks to the administrative and technical staff in Lund University for helping me out with various things throughout my PhD. In no particular order: **Elisabeth Nordström**, **Elisabeth Ohlsson**, **Linda Bienen**, **Margit Billesö**, **Erik Göthe**, **Erik Jonsson**, and **Stefan Moulnd**.

Richard Feynman, or rather the problem-solving algorithm that is attributed to him, has earned my salutations for teaching me how to solve problems. It goes like this: "Write down the problem, think very hard, write down the answer". Very useful. And, second-last but not least, it seems that giving thanks to your computer is becoming a thing these days [1], so I would like to send my regards[8] to **Zeffa**, my tireless Linux machine, who stuck with me all through these years and carried out my commands unquestionably. I will forgive you for randomly hanging on me on countless occasions, forcing me to restart by brute force. Such is life.

Finally, I take the opportunity here to thank my family: My sister **Foori** (**Farnoosh**) for being my big sis and for pushing me to go for PhD, my bro **Hamid** for being my little bro, my mom and dad **Goli** and **Papi G** (**Maryam** and **Mahmood**) for all the sacrifices they have made for us, my brother-in-law **Arash** for working hard and always being there to help, my auntie **Mariam Mortazavi** for being my Khaleii and for her boundless hospitality when I came to Sweden way back when, and my dear nephew **Radvin** for coming into this world, calling my D[9], and allowing me to let loose with my avuncular love.

MJ (Mohammad Attari)
Lund, October 2024

[7] Best Friend Forever.

[8] Not of the Lansiter variety.

[9] D for Dayi, which means uncle in Persian.

Contents

Preface

This thesis is the culmination of more than five years of work in the digital ASIC group at the *Electrical and Information Technology* department, Lund University, Sweden. The work was supervised by Professor *Liang Liu* (main supervisor) and Professor *Ove Edfors*. As I am not a person that is given to formality, I have striven[10] to keep the tone of the thesis light and friendly, in order to make it fun for the layperson as well as for the technically-oriented. Additionally, I have tried to follow the POLA[11] in writing the thesis, and I hope you enjoy reading it.

STRUCTURE OF THE THESIS

- **INTRODUCTION**
  The main body of the thesis consists of the publications appended in the back. The Introduction provides a broader and more comprehensive view than the very focused publications and ties their work together.

- **APPENDICES**

  A Code Examples
  Appendix A provides example C code for selected algorithms implemented in the ASIP.

- **PAPERS**
  The papers forming the main body of the thesis are reproduced in the back and listed in the following.

[10] Mayhap in vain ☹.
[11] Principle of least astonishment.

INCLUDED PAPERS

The following papers form the main body of this thesis and the respective published or draft versions are appended in the back.

Paper I: **Mohammad Attari**, Jesús Rodríguez Sánchez, Liang Liu, and Steffen Malkowsky, "An Application Specific Vector Processor for CNN-based Massive MIMO Positioning", *IEEE International Symposium on Circuits and Systems (ISCAS)*, May 2021, doi: 10.1109/ISCAS51556.2021.9401528.
Contribution: The main author, under the guidance of the contributing authors, designed an application-specific instruction set processor (ASIP) equipped with a CNN accelerator to provide an implementation for fingerprint-based positioning using massive MIMO technology. The processor was synthesized and evaluation results were provided.

Paper II: **Mohammad Attari**, Lucas Ferreira, Liang Liu, and Steffen Malkowsky, "An Application Specific Vector Processor for Efficient Massive MIMO Processing", *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 69, no. 9, pp. 3804–3815, Jun. 2022, doi: 10.1109/TCSI.2022.3182483.
Contribution: The main author, under the guidance of the contributing authors, was the designer of an ASIP, utilizing vector processing capabilities in the form of SIMD pipeline, VLIW architecture, and an 8×8 systolic array built into the pipeline, for efficient massive MIMO baseband processing. Additionally, the processor takes advantage of a parallel memory unit for easy access to matrix elements. The processor was synthesized in the 22nm process node and evaluation results were provided with comparison to the state-of-the-art.

Paper III: **Mohammad Attari**, Jesús Rodríguez Sánchez, and Liang Liu, "A Floating-Point 16×16 SVD Accelerator for Beyond-5G Large Intelligent Surfaces", *IEEE International Midwest Symposium on Circuits and Systems (MWSCAS)*, January 2024, doi: 10.1109/MWSCAS57524.2023.10406077.
Contribution: The main author, under the guidance of the contributing authors, designed and implemented an ASIC accelerator for SVD matrix decomposition, utilized in massive MIMO and large intelligent surfaces. The accelerator was customized for 8×8 and 16×16 matrices, and was synthesized and evaluation results were provided.

Paper IV: **Mohammad Attari**, Jesús Rodríguez Sánchez, Ove Edfors, and Liang Liu, "A 1095 pJ/b 219 Mb/s Application-specific Instruction-set Processor for Distributed Massive MIMO in 22FDX", *European Solid-State Electronics Research Conference - ESSERC*, 2024.

► *Paper accepted and presented.*

Contribution: The main author, under the guidance of the contributing authors, proposed an implementation for an ASIP processor trimmed for distributed massive MIMO systems. The processor comes equipped with two accelerators attached in the form of a 16 $\times$ 16 systolic array and a 16 $\times$ 16 SVD unit. The processor was designed, synthesized, placed-and-routed, verified, and finally taped out by the main author. A matching PCB was designed and the chip was successfully measured.

Paper V: **Mohammad Attari**, Ove Edfors, and Liang Liu, "Accelerator-assisted Floating-point ASIP for Communication and Positioning in Massive MIMO Systems",

► *Paper submitted to IEEE Transactions on Very Large Scale Integration (VLSI) Systems.*

Contribution: The main author, under the guidance of the contributing authors, was the sole designer of an ASIP geared for combined and efficient communication and positioning in a massive MIMO setup. The processor was designed with floating-point capabilities, replacing its fixed-point counterpart for parts of the processing. The processor was synthesized and evaluation results were provided.

Acronyms and Abbreviations

AI	Artificial Intelligence
AR	Augmented Reality
ASIC	Application-Specific Integrated Circuit
ASIP	Application-Specific Instruction-set Processor
BPS	Bits Per Second
BS	Base Station
CGRA	Coarse-Grained Reconfigurable Architecture
CISC	Complex Instruction Set Computer
CNN	Convolutional Neural Network
CPU	Central Processing Unit
CSI	Channel State Information
DMIMO	Distributed massive MIMO
DNN	Deep Neural Network
DSA	Domain-Specific Architecture
EDA	Electronic Design Automation

FFT	Fast Fourier Transform
FPGA	Field-Programmable Gate Array
FSM	Finite State Machine
GEMM	General Matrix Multiply
GK	Golub-Kahan
GPP	General Purpose Processor
GPU	Graphics Processing Unit
HDL	Hardware Description Language
IIC	Iterative Interference Cancellation
i.i.d.	independent and identically distributed
IoT	Internet of Things
ISA	Instruction Set Architecture
IUI	Inter-User Interference
KPI	Key Performance Indicator
LIS	Large Intelligent Surface
LTE	Long-Term Evolution
MAC	Multiply ACcumulate
MIMO	Multiple-Input Multiple-Output
ML	Machine Learning
MMSE	Minimum Mean Square Error
MR	Maximum Ratio
NN	Neural Network
NR	New Radio
OFDM	Orthogonal Frequency Division Multiplexing
OFDMA	Orthogonal Frequency Division Multiple Access
PE	Processing Element

QAM	Quadrature Amplitude Modulation
QRD	QR-Decomposition
RISC	Reduced Instruction Set Computer
RTL	Register Transfer Level
SIMD	Single Instruction Multiple Data
SIMT	Single Instruction Multiple Threads
SINR	Signal-to-Interference plus Noise Ratio
SISO	Single-Input Single-Output
SNR	Signal-to-Noise Ratio
SVD	Singular Value Decomposition
TDD	Time-Division Duplex
UE	User Equipment
UL	Up-Link
VR	Virtual Reality
ZF	Zero Forcing

Mathematical Notations

$\mathbb{C}$	Complex field
$\|\cdot\|_2$	ℓ_2-norm
$(\cdot)^*$	Complex conjugate
$(\cdot)^T$	Matrix/vector transpose
$(\cdot)^H$	Matrix/vector conjugate-transpose
$(\cdot)^{-1}$	Matrix inverse
$(\cdot)^\dagger$	Matrix pseudo-inverse
$(\cdot)_{ij}$	$(i,j)^{th}$ element of a matrix
$(\cdot)_{i:j:}$	Submatrix starting from row i and column j
$\mathcal{O}$	Order of computational complexity

INTRODUCTION

1

Motivation

This chapter serves as a stepping stone into the amazing world of communications, from the humanity's early attempts, all the way to the state-of-the-art systems that we have today, and looking forward to what we will have in the future. The ultimate purpose here is to throw some light on the problem of choosing an appropriate substrate as the main processing platform for the wireless communication systems of posterity.

1.1 THE BEGINNINGS

The primitive human communication started with speech and the development of the faculty of language approximately 100,000 years ago [2]. Then symbols were developed about 70,000 years later in the form of cave paintings. Petroglyphs, or rock carvings, were the next major advancement in communications that happened around the time the human beings became agrarian. From there we have pictograms (illustrations for ideas), ideograms (symbols for ideas), and finally scripts that turned into sophisticated writing systems. But this is not the whole story.

The concept of communicating something to others extends over to long distances as well (telecommunications). Smoke signals and drums were used thousands of years ago to accomplish the long-distance transfer of information. Physically moving a message from one location to another also goes back thousands of years. According to legend, Pheidippides, an Athenian herald, ran from a battlefield near Marathon to Athens, nonstop, to deliver the announcement of the Greek victory over the Persians in the

Battle of Marathon (490 BCE)[1]. Sadly, when he reached Athens he uttered a single phrase, "nikomen"[2], and then dropped dead on the spot to meet his maker. Pheidippides' story had an end, but the same can not be said about human ingenuity. From messenger birds being employed as early as 1350 BCE, if we fast forward to the 19th century, it only took us three millennia to advance science and technology to make electrical telecommunication systems possible, with the very first telegram sent by Samuel Morse across a 3-km length of wire on 11 January 1838. This opened the floodgates of telecommunications, and it is hard to imagine that anything can put a stop to the deluge of innovation.

1.2 THE EXPLOITATION OF LIMITED RESOURCES

But like many other things in our world, in this field we are also dealing with limited resources. Based on Claude Shannon's discovery [3], a communications system's theoretical upper bound on the achieved data rate can be expressed as

$$C = B\ log_2(1 + SINR), \tag{1.1}$$

in which C is the channel capacity measured in Bits Per Second (BPS), B denotes the bandwidth of the system expressed in Hz, and Signal-to-Interference plus Noise Ratio (SINR) encompasses the receiver's acquired signal power, divided by the power due to noise and interference. What Equation 1.1 implies is that in order to improve a communication system's capacity, and by extension its data rate, one needs to either increase the bandwidth or crank up the signal power. The latter is the less attractive, or rather less feasible, alternative due to its logarithmic nature and practical limits on power delivery. Therefore, we are left with bumping up the bandwidth as the main method of capacity improvement, as evidenced by the widening of carrier bandwidth in wireless standards over generations (from 200 kHz in 2G all the way to 100-400 MHz in 5G and 6G [4, 5]). As spectrum is a key connectivity asset, multiple-antenna techniques, such as Multiple-Input Multiple-Output (MIMO), have gained popularity as an important method in improving spectral efficiency. Systems in 4G use 2×2 and 4×4 MIMO, while in 5G massive MIMO systems featuring up to 200 antennas are possible. The expectation is that 6G will scale up the bandwidth to 400 MHz, and will put up to 1024 antenna elements to use [6].

[1] As the author is Persian, he's not too thrilled about this little anecdote, and he advises the readers to take the whole story with a huge grain of salt.

[2] We win!

All of these crazy numbers mean that we need processing systems that do not shy away when it comes to the crazy factor. In the next section I will look into this issue, and try to provide guidelines to help us ascend safely to the processing summit.

1.3 THE GUIDELINES

The fast-changing nature of standards and requirements favors flexibility, while the exorbitant user demands in terms of traffic call for customized systems. This era has been heralded as the golden age of computer architecture [7], and with good reason. The general-purpose systems are being replaced by Domain-Specific Architectures (DSAs), that tackle a narrower range of tasks but in an extremely efficient manner. This is by no means a license to use DSAs haphazardly. There must be some order to the madness, and that is why we set up guidelines to aid us in our decisions. These DSA guidelines are as follows [8]:

1. Reduce data movement and shuffling by the use of dedicated, less energy-hungry, memories (e.g. software-controlled scratchpads and custom storage idioms [9]).
2. Favor arithmetic muscle over beefy pipeline improvements, for instance by using more number of simpler cores.
3. Exploit the inherent parallelism in the target domain by exposing it in the programming model (e.g. using Single Instruction Multiple Data (SIMD)).
4. Use custom data types and smaller data sizes that fit the domain (e.g. use bfloat16 [10] for machine learning).
5. Use a domain-specific programming language for easier application porting to the DSA (e.g. Halide [11] for vision processing and Tesnorflow [12] for Deep Neural Networks (DNNs)).

I will come back to these guidelines in chapter 4, where microarchitecural discussions take place.

I would be remiss if I did not address the elephant in the room, and yes, you guessed right, I mean Artificial Intelligence (AI) and Machine Learning (ML). It is no insignificant feat when AlphaGo had the honor of being the first computer to beat humans in the ancient game of Go with no handicap [13] in 2015, while the ship had sailed long before that for the game of chess in 1997, when IBM's Deep Blue defeated the reigning world champion Garry Kasparov. Putting games aside, the communications field is no exception, and

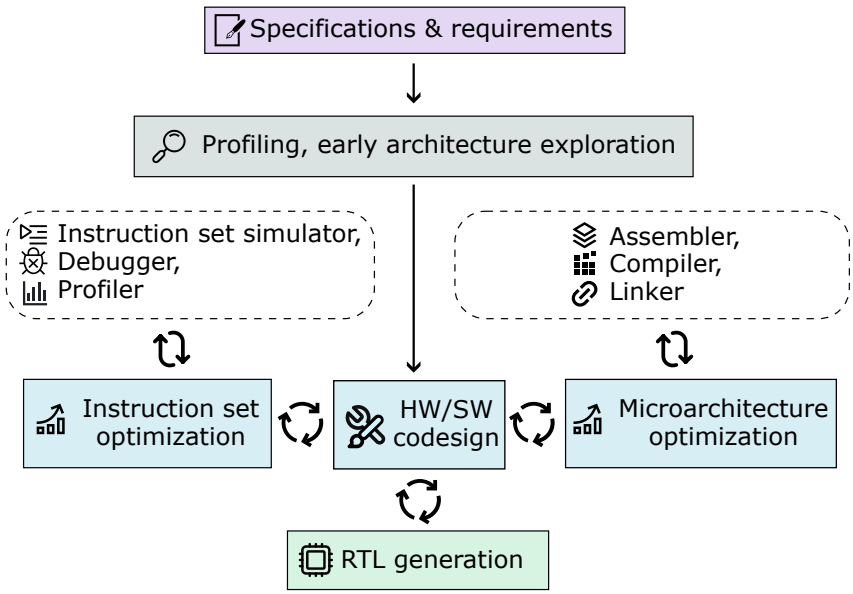

Figure 1.1: The hardware-software co-design methodology in ASIP flow.

can join the dark side[3]. For instance, ML can be used in a communications system to help in tasks such as positioning.

1.4 THE CO-DESIGN METHODOLOGY

The list of guidelines, codified above, operates as an overarching compass when it comes to design decisions for DSAs, but it does not specify the methodology to follow from specifications and requirements to final implementation. The hardware-software co-design methodology tries to design the software and hardware components concurrently in order to leverage the synergy between the duo [14]. This philosophy is embodied in the development of processors customized for specific applications, which are more widely known as Application-Specific Instruction-set Processors (ASIPs) [15]. Figure 1.1 diagrams the steps involved in the ASIP co-design flow. In this flow, the tools aid the designer in gradually specifying and optimizing the instruction set by keeping the compiler in the loop. The dashed boxes in the figure portray the automatic generation of the toolchain (i.e. assembler, compiler, and linker), coupled with the useful accoutrements for simulation (that is,

[3]"If you can't beat us ... join us", Agent Smiths in The Matrix Reloaded (2003).

instruction set simulator, debugger, and profiler). This tight interaction eases the process of hardware/software partitioning and optimization. For instance, matrix multiplication can be implemented and profiled using a simple design first. Then, if the performance is not satisfactory, the processor pipeline can be refined (e.g. by adding SIMD lanes) to make it more adept at parallel computation. This iterative improvement can be even pushed further by offloading the matrix multiplication in its entirety to a dedicated unit, such as a systolic array, if desired.

1.5 THE CONTRIBUTION

Creating systems that are both performant and efficient in handling the requirements of the wireless technology of the future is going to be a tall order. In this thesis I will first present the requirements, and then address what is needed to design systems that meet those requirements in the volatile field of communications. The content is chiefly based on five papers that put the spotlight on algorithm-hardware co-design using ASIPs that work in concert with accelerators. These papers are listed below in chronological order:

- **Paper I** kicks things off with an implementation for a fingerprint-based positioning processor in the context of a massive MIMO system. A configurable Convolutional Neural Network (CNN) accelerator is paired up with an ASIP to utilize the information obtained from the wireless channel for localization purposes.
- **Paper II** untangles some of the linear algorithms that are commonly encountered in digital massive MIMO processing. The algorithms are dissected to tease out their operational profiles, and an ASIP is co-designed to efficiently implement these algorithms. Additionally, accelerator integration is introduced in places that the ASIP misses the mark on performance, with the proviso that this integration conforms to the philosophy of programmer-visibility.
- **Paper III** steps away from ASIPs momentarily to propose an Application-Specific Integrated Circuit (ASIC) implementation for the holy grail of matrix decompositions, i.e. the honorable Singular Value Decomposition (SVD). The accelerator is designed specifically for distributed and scalable massive MIMO systems and Large Intelligent Surfaces (LISs), and supports 32-bit floating-point representation for a wide dynamic range.
- In **Paper IV**, the SVD accelerator presented in Paper III is integrated into an ASIP, along with a 16×16 systolic array and a parallel memory module, to be used in a Distributed massive MIMO (DMIMO) system. The C-programmable processor is taped out and measured.

- Finally, **Paper V** introduces a mixed-datatype ASIP, suitable for both positioning and communication scenarios. The system utilizes a beefed-up systolic array to support a variety of General Matrix Multiply (GEMM) sizes efficiently, and it also features a CNN accelerator with its own memory for the positioning task.

2

Wireless Communications Principles

This chapter touches upon the modern wireless communications principles, introducing the concepts behind Single-Input Single-Output (SISO), MIMO, massive MIMO, distributed massive MIMO and Orthogonal Frequency Division Multiplexing (OFDM).

2.1 TECHNIQUES

2.1.1 SISO

Before we get into the nitty-gritty details of a complicated system, let us first consider the simple case of a narrowband system with a single transmit antenna and a single receive antenna, also referred to as a SISO system, as depicted in Figure 2.1. This system can be modeled[1] by

$$y = hs + n, \tag{2.1}$$

in which s is the complex-valued transmitted signal, y the received signal, h the channel gain (impulse response) denoting attenuation and phase shift

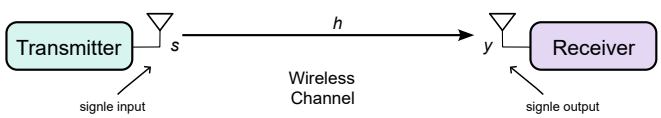

Figure 2.1: A SISO system.

[1] Keeping in mind that: "All models are wrong, some are useful" [16].

introduced by the air interface, and n is the noise[2]. Now, assuming the receiver has full knowledge of the channel (that is, it knows h), performing detection[3] can be simply done by

$$s = y/h - n/h. \tag{2.2}$$

A SISO system feels right at home when its channel Signal-to-Noise Ratio (SNR) is high.

2.1.2 MIMO

Current wireless systems employ a technique known as MIMO by utilizing antenna arrays, both at the input and output, that can exploit the spatial domain so as to increase the robustness and throughput of the radio link. With spatial multiplexing multiple antennas can be utilized (both in the transmitter and receiver) in order to deliver many streams of traffic on the same time-frequency resource. The word *input* in MIMO refers to the transmit antennas, while receive antennas act as outputs to the air interface. Figure 2.2 shows a MIMO system with M_T transmit and M_R receive antennas, or an $M_R \times M_T$ MIMO system.

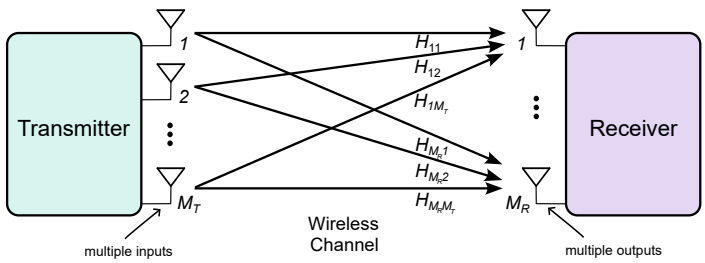

Figure 2.2: A MIMO system.

The fundamental concepts related to the MIMO technology can boil down to *spatial diversity*, *spatial multiplexing*, and *beamforming*. The first two concepts are diagrammed in Figure 2.3 [17]. Spatial diversity at the transmitter means that the same signal stream is sent over more than one antenna, increasing the chance that the signal will arrive successfully at the target without increasing the data rate. Spatial multiplexing, on the other hand, sends multiple data streams from a number of antennas, hoping to gain on data rate while

[2]Additive independent and identically distributed (i.i.d.) circularly symmetric complex Gaussian noise, to be more exact.

[3]Technically, this is estimation, which must be followed by symbol selection.

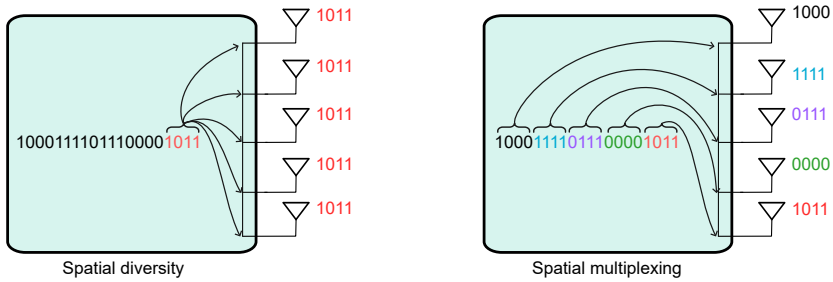

Figure 2.3: Spatial diversity vs. spatial multiplexing.

ignoring the diversity gain. Beamforming concerns itself with steering the main lobe of the beam towards a target of interest, while possibly trying to avoid absorbing/delivering interference from/to the other devices. Figure 2.4 shows the concept of beamforming, in a simplified example, from the point of view of the Base Station (BS) transmitting to a particular target [18]. In part (a) of the figure Zero Forcing (ZF) is used so the interferers are placed within the nulls to completely suppress interference, but failing to align the main lobe on the target[4]. In (b), the Maximum Ratio (MR) receiver is placed exactly in the center of the main lobe, with the downside of ignoring the contribution from interferers. Finally, in part (c) Minimum Mean Square Error (MMSE) is used as a compromise between interference cancellation and signal-to-noise ratio.

The narrowband MIMO system is characterized by

$$\boldsymbol{y} = \boldsymbol{H}\boldsymbol{s} + \boldsymbol{n}, \tag{2.3}$$

where $\boldsymbol{H}$ is the complex-valued $M_R \times M_T$ channel matrix, $\boldsymbol{s}$ the $M_T \times 1$ transmit signal vector, $\boldsymbol{y}$ the $M_R \times 1$ received signal, and n the noise vector. For simplification purposes, the complex $\boldsymbol{H}$ matrix is assumed to be unchanging over a small period of time (shorter than the coherence time) in a slow fading channel. In linear detection methods, after estimating the channel matrix, the channel effect is compensated by another matrix. The recovery process of the transmitted vector $\boldsymbol{s}$ from the received signal $\boldsymbol{y}$ is more complicated than the SISO case discussed previously, as it now involves operations such as matrix inversion. A simple solution to recover the transmit vector is through inverting the channel matrix, given by

$$\boldsymbol{s} = \boldsymbol{H}^{-1}\boldsymbol{y} - \boldsymbol{H}^{-1}\boldsymbol{n} = \boldsymbol{H}^{-1}\boldsymbol{y} + \boldsymbol{n}'. \tag{2.4}$$

[4]For the uninitiated, the long tube-looking thing is the main lobe, and the smaller ones are the side lobes.

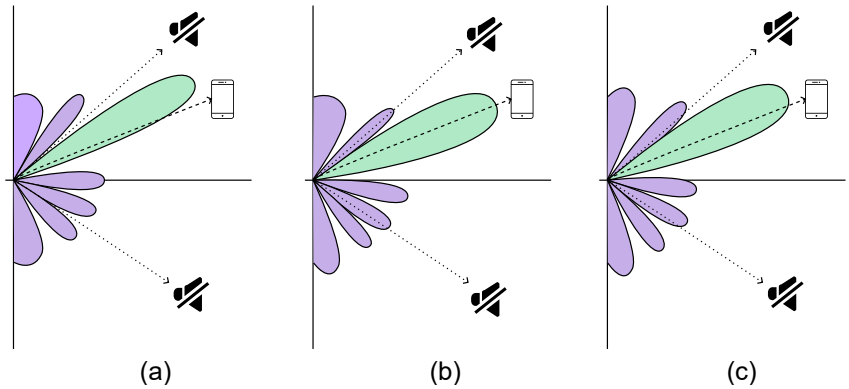

Figure 2.4: Beamforming, using (a) zero forcing, (b) maximum ratio, and (c) minimum mean square error. The target is depicted with a cell-phone and interferers with a crossed-over megaphone.

But since Equation (2.4) requires matrix inversion[5], an alternative is to approximate the desired signal with a least-squares solution according to

$$\boldsymbol{s} = \boldsymbol{H}^{\dagger}\boldsymbol{y} + \boldsymbol{n}'', \tag{2.5}$$

where $\boldsymbol{H}^{\dagger}$ represents the pseudo-inverse of the channel matrix, computed as

$$\boldsymbol{H}^{\dagger} = (\boldsymbol{H}^{H}\boldsymbol{H})^{-1}\boldsymbol{H}^{H}. \tag{2.6}$$

There are also non-linear detection methods, e.g. sphere decoding, that might become necessary in very demanding channels.

2.1.3 MULTI-USER MIMO

The MIMO technology can be applied to cases where multiple users are involved in a communication system. Figure 2.5 shows a simple example system with $M = 3$ BS antennas serving $K = 2$ users simultaneously. The channel matrix here is represented with a $K \times M$, or 2×3, matrix $\boldsymbol{H}$.

In this case, as each user is not privy to what the other one is receiving, the BS has to bear the weight of processing by employing a precoding matrix $\mathbf{P}$, which is applied to the symbols to compute the transmitted signals with

$$\boldsymbol{x} = \boldsymbol{P}\boldsymbol{s}. \tag{2.7}$$

[5]Well, OK, nothing wrong with inversion per se, but it is a costly procedure and, on top of that, requires an invertible matrix. Boo!

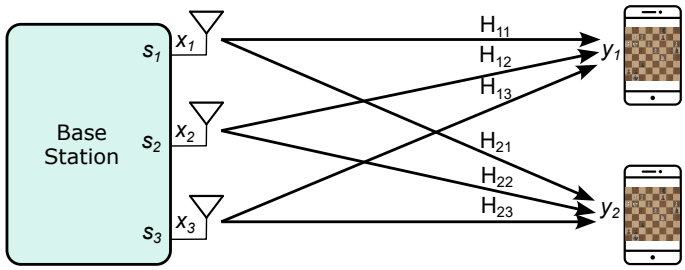

Figure 2.5: A multi-user MIMO system.

Depending on how well-conditioned the channel matrix is and the noise level, the BS can use a ZF, an MR, or an MMSE precoder according to the following equations, respectively

$$\boldsymbol{P}_{ZF} = \boldsymbol{H}^H(\boldsymbol{H}\boldsymbol{H}^H)^{-1}, \tag{2.8}$$

$$\boldsymbol{P}_{MR} = \boldsymbol{H}^H, \tag{2.9}$$

$$\boldsymbol{P}_{MMSE} = \boldsymbol{H}^H(\boldsymbol{H}\boldsymbol{H}^H + \beta\boldsymbol{I}_K)^{-1}. \tag{2.10}$$

If interference cancellation is the prime target, then the BS will choose ZF in order to place the interferers precisely within the nulls. On the other hand, if the channel matrix is singular or the noise is large, the BS will place its bets on MR. Finally, MMSE serves as a compromise, by taking the middle ground between noise enhancement and interference cancellation.

2.1.4 MASSIVE MIMO

It is not exactly a new concept to up the ante on spatial-domain utilization by increasing the number of antennas in the BS [19]. The idea is that by scaling the number of antenna elements towards infinity (Figure 2.6), and assuming ideal conditions, you can reap the beneficial properties by adding the further exploitation of the spatial domain to the mix [20–23]. The huge number of antennas in the array provides a high spatial resolution, resulting in precise signal steering toward the User Equipments (UEs). By operating in Time-Division Duplex (TDD) mode and exploiting the radio channel reciprocity, the BS can combine beamforming and precoding to focus the transmitted signals in a specific direction.

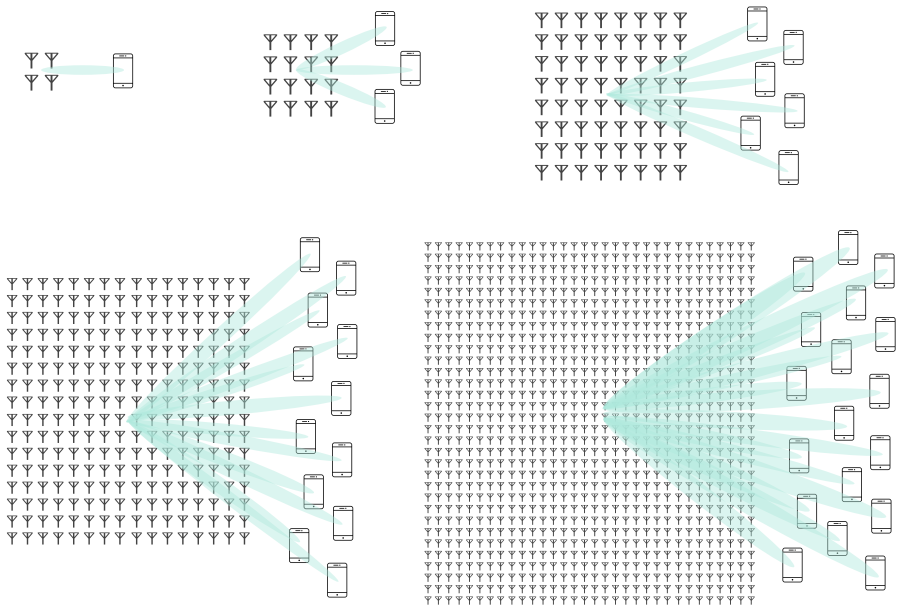

Figure 2.6: Massive MIMO.

In conventional systems the signals are sent out in all directions, resulting in Inter-User Interference (IUI). In massive MIMO the signals undergo precoding in the BS, using the information contained in the Channel State Information (CSI), in order to make them add up constructively in the intended UEs. Figure 2.7 demonstrates this concept in a sparse multipath environment. The transmission of multiple streams of data simultaneously is highly attractive when the resources are in short supply, but it has the downside of requiring complex signal processing to separate the data streams. This processing is typified in operations such as matrix inversion, QR decomposition, Cholesky decomposition, SVD, etc, which are dealt with in the next chapter.

2.1.5 DISTRIBUTED MASSIVE MIMO

Up to this point when I was talking about massive MIMO I meant collocated architectures, where all the antennas are located close to each other in order to reduce the backhaul requirements. To obtain higher spectral efficiency and improved coverage, a massive MIMO system can be deployed in a distributed fashion, with the service antennas spread out over an area. This has the negative impact of having to deal with increased backhaul requirements. Furthermore, new user experiences such as self-generated content, video

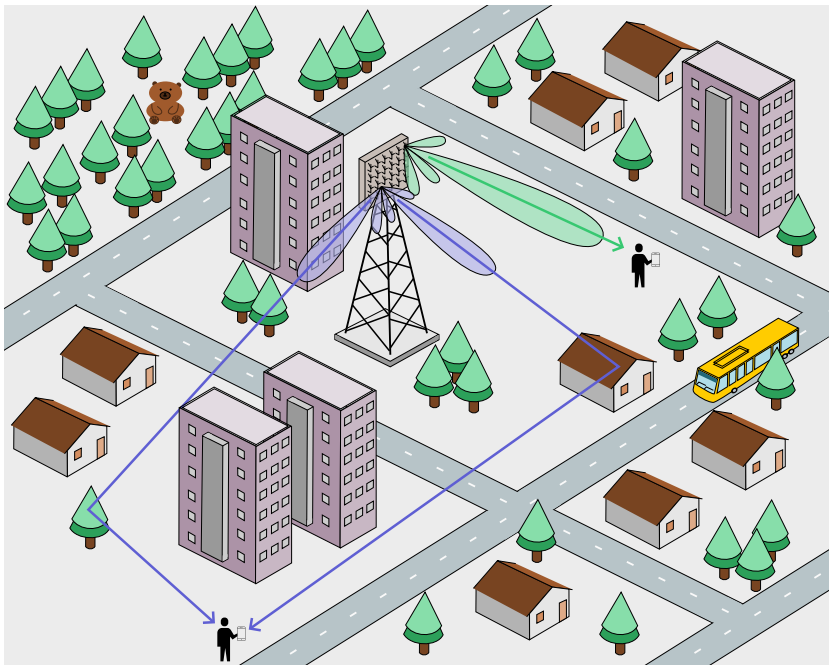

Figure 2.7: Beamforming and precoding in massive MIMO in a multipath environment.

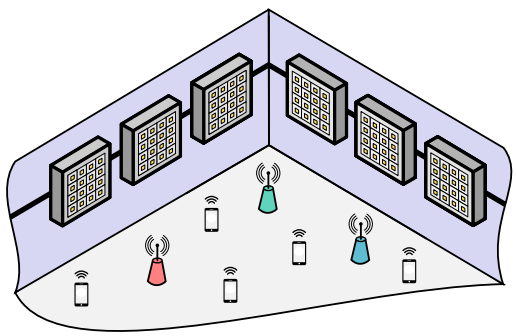

Figure 2.8: Distributed massive MIMO.

conferencing, etc. mean that wireless systems are seeing an increase in uplink utilization. Mobile networks today are highly asymmetric, with the downlink direction having much faster speed than the uplink. Distributed baseband processing is an appealing solution to this challenge.

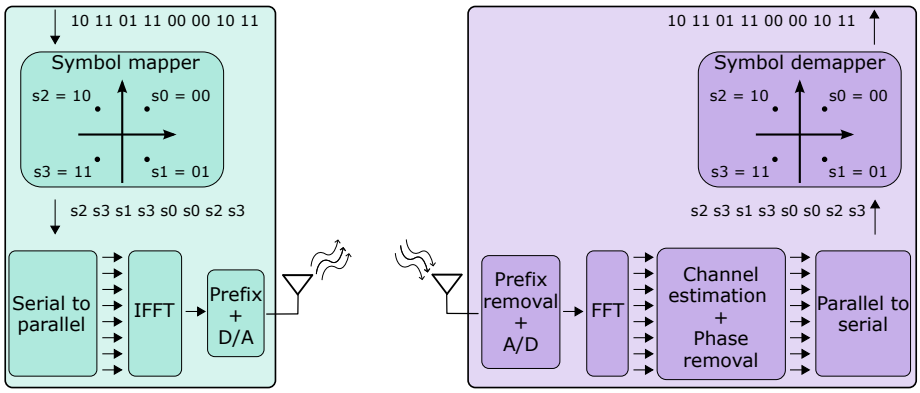

Figure 2.9: An OFDM transmitter (left) and receiver (right).

Scaling up the number of antenna elements even further is the name of the game, when it comes to systems such as LISs [24]. But due to the challenges presented by central processing limits, these systems often opt for a more distributed architecture.

2.1.6 OFDM

One way of combating the frequency selectivity phenomenon, without increasing the bandwidth, is through a technique that has withstood the test of time known as OFDM [25,26]. In OFDM the bandwidth is divided into a series of orthogonal subcarriers, obviating the need for inter-subcarrier guard bands, and easing equalization. Each subcarrier, therefore, has a small bandwidth and low symbol rate, but all the subcarriers, put together, have the same bandwidth of a wideband single-carrier system. Figure 2.9 sketches an OFDM transmitter/receiver pair. OFDM has been so successful that it is now the go-to modulation scheme in modern wireless communication systems (that is from 4G Long-Term Evolution (LTE) onward).

The simplicity of OFDM, coupled with the fact that time-frequency data conversion can take advantage of the computationally efficient Fast Fourier Transform (FFT) and its inverse, led to it being picked up for 4G, as well as 5G and beyond. Figure 2.10 outlines how OFDM can be adapted to allow interacting with multiple users' data streams, which in communications parlance is dubbed as Orthogonal Frequency Division Multiple Access (OFDMA).

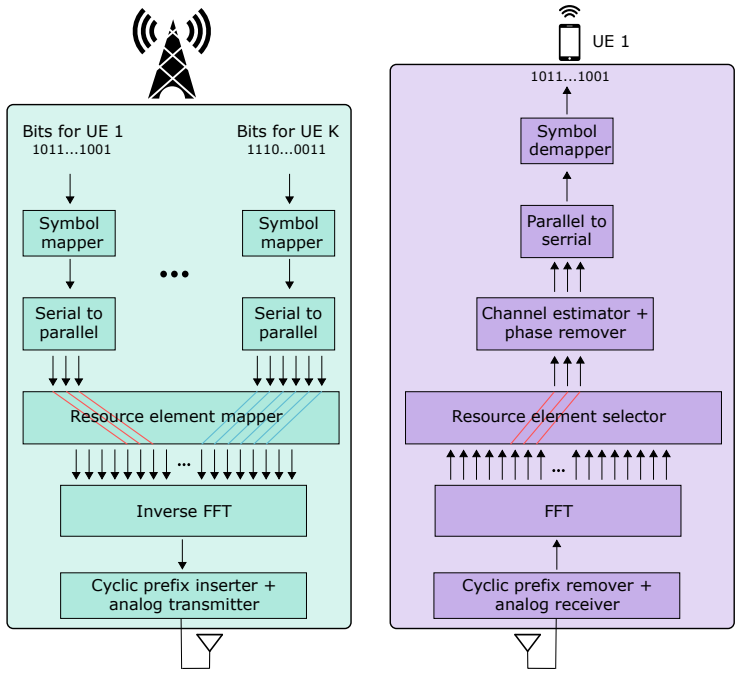

Figure 2.10: A BS transmitting to multiple UEs using OFDMA (only one UE is shown on the right).

2.2 APPLICATIONS AND REQUIREMENTS

Wireless connectivity is going to be key in driving future applications in industrial, entertainment, healthcare, and sport sectors. The environments in which this connectivity will play a significant role include, but are not limited to, robot-operated factories, large stadiums, hospitals, nursing homes, disaster areas, and smart homes. The list of applications is endless, but the use cases can be shoehorned into three main categories [27]:

- Monitoring and real-time applications.
- Augmented Reality (AR)/Virtual Reality (VR) applications.
- Location-based applications.

The typical use cases are diagrammed in the tri-hexagonal style of Figure 2.11. Each individual use case has its own requirements, which are tabulated in Table 2.1 in terms of the required data rate, accuracy and end-to-end latency. The 5G-and-beyond standards need to be up to par with these requirements, and by inspecting the 5G Key Performance Indicators (KPIs) we can observe that they promise peak data rates of 20 Gbit/s and

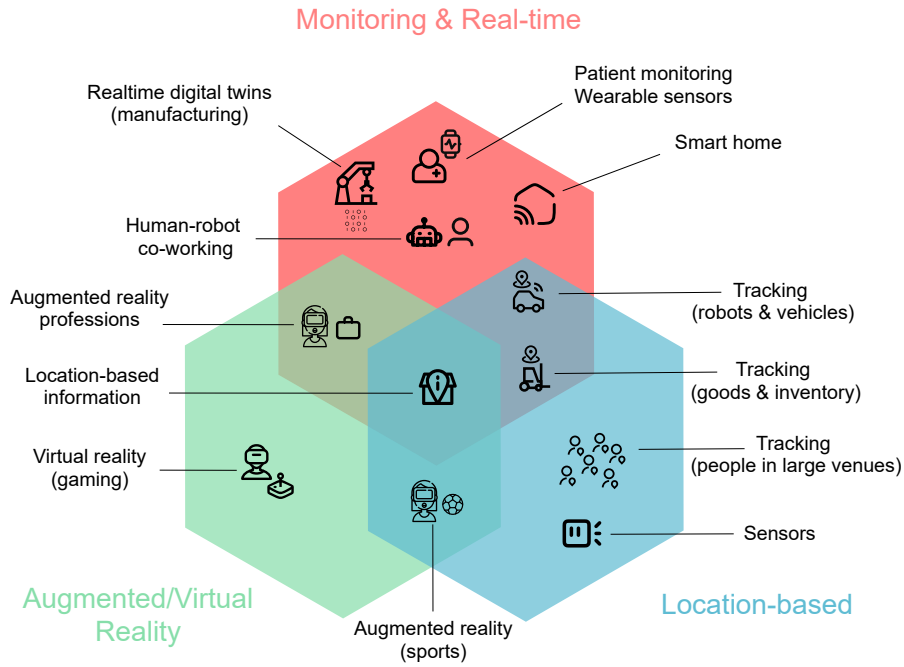

Figure 2.11: Use cases.

a perceived user-experienced data rate of 100 Mbit/s [5]. While these are pretty impressive numbers on their own, the 6G standard is not one to be outshined in this regard, and as a result, the peak data rate in 6G aims to reach 1000 Gbit/s, with users predicted to experience 1 Gbit/s speeds. Figure 2.12 employs a spider-web diagram to illustrate the 6G KPIs and pits them against those of the 5G standard [28].

These requirements imply that the communication system's infrastructure must be able to support localization and sensing, with centimeter-level precision. New scenarios for end user equipments, including ultra low power and battery-less devices, gesture-activated interfaces, and network-on-device units are to be expected. Furthermore, adaptation to new network conditions with AI/ML techniques will become the norm [29].

Now, let us see how the OFDM-based resource grid, which is in common use in wireless communications systems, ties into this requirements discussion. I start by inspecting the typical frame structure in 5G New Radio (NR), as outlined in Figure 2.13 [5]. One frame with a duration of $T_{frame} = 10$ ms is divided into ten 1 ms subframes, each consisting of one slot of 14 OFDM symbols in this case ($\Delta f = 15$ kHz numerology). The allotted precoding time

Table 2.1: Use cases and their requirements [27].

Requirements			
Use case	**User data rate**	**Positioning accuracy**	**Latency**
Augmented reality (sports)	< 5 Mbps	< 0.5 m	< 20 ms
Augmented reality (professions)	< 45 Mbps (compressed) < 3 Gbps (uncompressed)	< 0.1 m	< 10 ms
Virtual reality (gaming)	< 150 Mbps	< 0.1 m	< 1 ms (control) < 100 ms (data)
Realtime digital twins (manufacturing)	1 Mbps	0.1 m	< 1 ms - 50 ms
Patient monitoring (wearables)	100 bps - 1 Mbps	< 1 m	< 200 ms
Human-robot co-working	5 Mbps	–	1 ms
Tracking (goods & inventory)	< 1 Mbps	< 0.1 m	< 100 ms - 1 s
Tracking (people in large venues)	< 1 kbps (positioning) > 1 Mbps (navigation)	< 1 m	< 1 s
Tracking (Robots & vehicles)	< 10 Mbps	< 0.1 m	< 10 ms
Smart home	< 50 kbps	< 0.1 m	< 100 ms

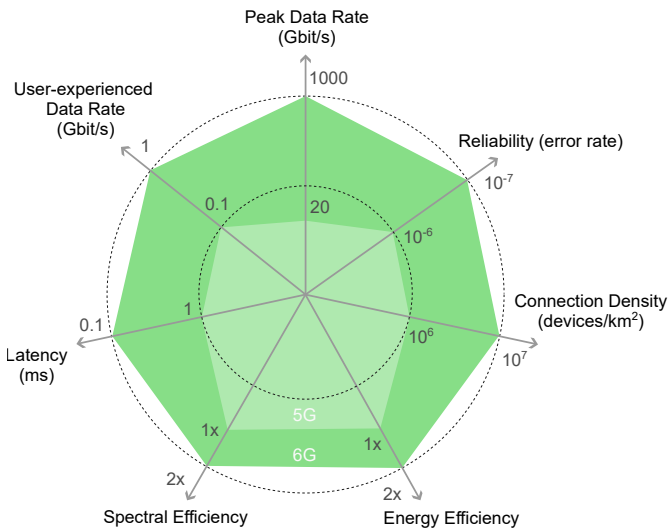

Figure 2.12: 6G vs. 5G key performance indicators.

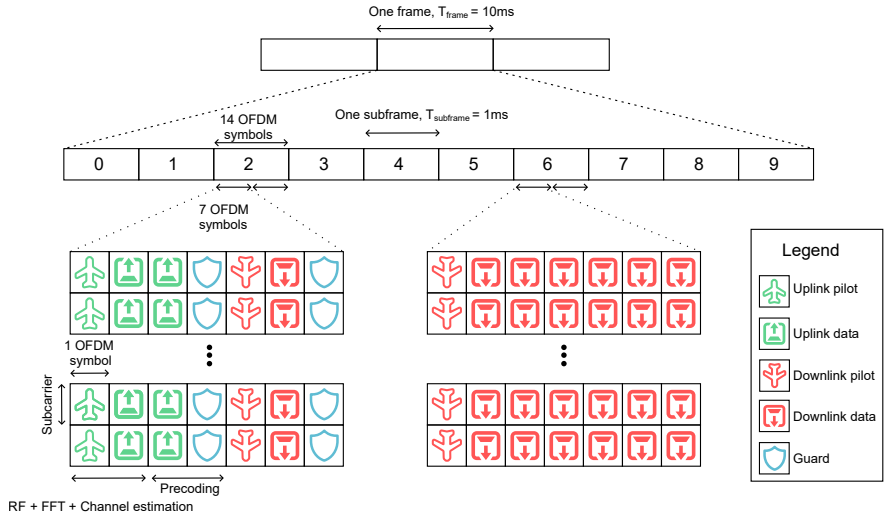

Figure 2.13: Example frame structure in 5G NR.

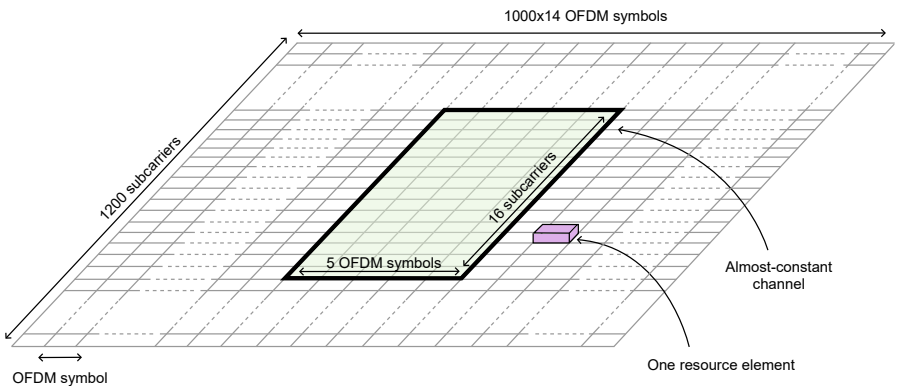

Figure 2.14: One-second snapshot of the resource grid for a 20 MHz bandwidth channel with 1200 15-kHz subcarriers and 14000 OFDM symbols.

for the stringent case shown in the left is only around two OFDM symbols, or just under 150 µs, after the Up-Link (UL) pilots have been received and FFT and channel estimation are complete.

Figure 2.14 sketches the time-frequency grid for a 20 MHz channel bandwidth in a one-second period. The resource grid in the time domain consists of a 100 frames, which translates to 14,000 OFDM symbols. For this system in a 128 $\times$ 16 massive MIMO setting, and with a 15 kHz subcarrier spacing,

Table 2.2: Required inversions per second (with the same coherency assumptions for all cases).

Subcarrier spacing	Bandwidth	#Subcarriers	#OFDM symbols in one second	#Resource elements in one second	#Inversions/s
15 kHz	20 MHz	1200	14000	16.8M	210k
15 kHz	50 MHz	3300	14000	46.2M	580k
30 kHz	100 MHz	3300	28000	92.4M	1.16M
60 kHz	100 MHz	1650	56000	92.4M	1.16M
60 kHz	200 MHz	3300	56000	184.8M	2.3M

there will be 1200 subcarriers. This means there exist a total of 1200 $\times$ 14000, or 16.8 M, resource elements[6] in one second. Assuming the channel remains almost constant over 16 subcarriers and 5 OFDM symbols, as visualized with the highlighted block in the figure, 75 $\times$ 2800 or 210k channel inversions need to performed every second. For the more extreme case of a 100 MHz channel bandwidth with 60 kHz subcarrier spacing, there are 1650 subcarriers and 56000 OFDM symbols in one second, which amounts to a total number of 92.4 M resource elements, which in turn leads to approximately 1.16M channel inversions/s. This is further elaborated in Table 2.2, which explores more scenarios.

To sum up, here I considered the requirements from the standpoint of applications and the whole system. In the following chapter I will switch to second gear by putting a premium on the algorithms that show up in communications processing, and consider the requirements form their vantage point.

[6] A resource element is the smallest time-frequency unit of one OFDM symbol and one subcarrier.

3

Digital Signal Processing and Algorithms

In his chapter I will get into the meat of algorithms that crop up repeatedly in digital signal processing for communications, positioning, and matrix decompositions. Now, onto details!

3.1 MASSIVE MIMO ALGORITHMS

For the discussions here I consider an OFDM-based massive MIMO system with linear algorithms. Under favorable channel conditions, the massive number of antennas allows linear algorithms to approach the performance of their non-linear counterparts [21]. For the ZF algorithm, the relevant part of the channel state matrix, $\boldsymbol{H}$, is used to perform symbol detection (a.k.a. postcoding). The communication algorithm for a multi-user MIMO system, comprised of M antenna elements and K UEs, can be summarized as follows. An estimate for the transmitted symbol vector $\hat{\boldsymbol{y}} \in \mathbb{C}$ is acquired by multiplying the $M \times 1$ received data vector $\boldsymbol{r}$ by the $K \times M$ detection matrix $\boldsymbol{W}_{\text{det}}$, expressed as

$$\hat{\boldsymbol{y}} = \boldsymbol{W}_{\text{det}} \boldsymbol{r}. \tag{3.1}$$

The detection matrix $\boldsymbol{W}_{\text{det}}$ in Equation (3.1) is obtained from the information contained in the estimated UL channel matrix $\boldsymbol{H}$. For the ZF algorithm, the $\boldsymbol{W}_{\text{det}}$ is equivalent to the pseudo-inverse of the estimated channel matrix $\boldsymbol{H}^{\dagger}$, couched in mathematical terms as

$$\boldsymbol{W}_{\text{det}} = \boldsymbol{H}^{\dagger} = (\boldsymbol{H}^{\mathsf{H}} \boldsymbol{H})^{-1} \boldsymbol{H}^{\mathsf{H}}. \tag{3.2}$$

$$\begin{bmatrix} m_{00} & m_{01} & m_{02} \\ m_{10} & m_{11} & m_{12} \\ m_{20} & m_{21} & m_{22} \end{bmatrix} \times \begin{bmatrix} v_0 \\ v_1 \\ v_2 \end{bmatrix} = \begin{bmatrix} m_{00} \\ m_{10} \\ m_{20} \end{bmatrix} \times v_0 + \begin{bmatrix} m_{01} \\ m_{11} \\ m_{21} \end{bmatrix} \times v_1 + \begin{bmatrix} m_{02} \\ m_{12} \\ m_{22} \end{bmatrix} \times v_2$$

Figure 3.1: A 3×3 by 3×1 matrix-vector multiplication example.

A mere inspection of Equation (3.2) reveals that the process involves the calculation of the Gramian matrix ($\boldsymbol{H}^{\mathsf{H}}\boldsymbol{H}$), followed by its inversion and a subsequent multiplication with $\boldsymbol{H}^{\mathsf{H}}$. Finally, the detection is accomplished in equation (3.1) by a final multiplication with the received data vector $\boldsymbol{r}$. This means three matrix operations are involved:

- Matrix-vector multiplication
- Matrix-matrix multiplication
- Matrix inversion

The subsections below will try to take a sneak peek and look at these operations in more detail.

3.1.1 MATRIX-VECTOR MULTIPLICATION

Matrix-vector multiplication can be viewed either as a number of vector dot products or, alternatively, as the sum of a number of scalar-vector products. Figure 3.1 depicts the latter for a 3×3 matrix multiplied by a 3×1 vector. In this case the matrix-vector multiplication turns into three vector-scalar multiplications, and two vector additions.[1]

3.1.2 MATRIX-MATRIX MULTIPLICATION

Matrix-matrix multiplication follows the same concept discussed previously, except that it involves multiple matrix-vector multiplications. Figure 3.2 portrays an example for a 3×3 matrix multiplied by a 3×2 matrix. Here, the matrix-matrix multiplication is equivalent to six vector-scalar multiplications and 4 vector additions. Both methods mesh well with a hardware system that has vector-based capabilities.

3.1.3 MATRIX INVERSION

Matrix inversion is a time-consuming operation, and thereby decompositions are used to speed up the process. Typical examples are QR-Decomposition (QRD) [31], extended QRD, and Cholesky decomposition [32], accompanied

[1] You might not like the numbering system starting from 0, but I only opted to stand on the shoulders of giants [30].

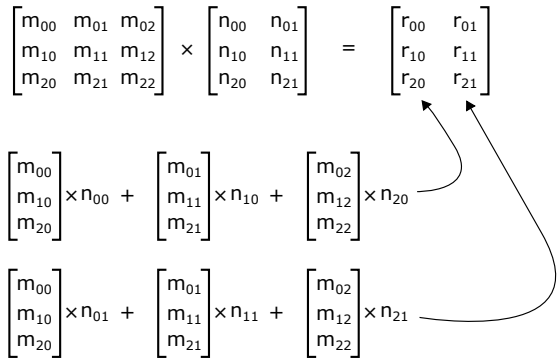

Figure 3.2: A 3×3 by 3×2 matrix-matrix multiplication example.

Algorithm 1 Modified Gram-Schmidt QRD

Input: $K \times K$ complex matrix $\boldsymbol{A}$
Output: Unitary matrix $\boldsymbol{Q}$ and upper triangular matrix $\boldsymbol{R}$

$\boldsymbol{Q} \leftarrow \boldsymbol{0}_K$
$\boldsymbol{R} \leftarrow \boldsymbol{0}_K$
$\boldsymbol{V} \leftarrow \boldsymbol{A}$
for $i \leftarrow 1$ **to** K **do**
    $r_{ii} \leftarrow \|\boldsymbol{V}(:,i)\|_2$
    $\boldsymbol{Q}(:,i) \leftarrow \boldsymbol{V}(:,i)/r_{ii}$
    for $j \leftarrow i+1$ **to** K **do**
        $r_{ij} \leftarrow \boldsymbol{Q}(:,i) \cdot \boldsymbol{V}(:,j)$
        $\boldsymbol{V}(:,j) \leftarrow \boldsymbol{V}(:,j) - r_{ij}\boldsymbol{Q}(:,i)$
    end for
end for

with backward- and/or forward substitutions. Depending on the different number of users and antennas one of these factorization methods can be used.

3.1.4 QRD

The QRD algorithm can be used to decompose a complex square matrix $\boldsymbol{A}$ into an upper triangular matrix $\boldsymbol{R}$, and a unitary matrix $\boldsymbol{Q}$ [33]. The mathematical representation of this is expressed as

$$\boldsymbol{A} = \boldsymbol{Q}\boldsymbol{R}. \tag{3.3}$$

Given that $\boldsymbol{Q}$ is a unitary matrix ($\boldsymbol{Q}^H\boldsymbol{Q} = \boldsymbol{I}$), we can see that $\boldsymbol{Q}$'s conjugate transpose (i.e. its Hermitian) is equivalent to its inverse, hence for $\boldsymbol{Q}$ we have

Algorithm 2 Back substitution

Input: $K \times K$ upper-triangular matrix $\boldsymbol{A}$
Output: $\boldsymbol{B}$ as the solution to $\boldsymbol{AB} = \boldsymbol{I}_K$
1: $\boldsymbol{B} \leftarrow \boldsymbol{I}_K$
2: **for** $i \leftarrow K$ **to** 1 **do**
3: **for** $j \leftarrow i + 1$ **to** K **do**
4: $\boldsymbol{b}_i \leftarrow \boldsymbol{b}_i - a_{ij}\boldsymbol{b}_j$
5: **end for**
6: $\boldsymbol{b}_i \leftarrow \boldsymbol{b}_i / a_{ii}$
7: **end for**

$\boldsymbol{Q}^{-1} = \boldsymbol{Q}^{\mathsf{H}}$, and the inverse of $\boldsymbol{A}$ is now simply acquired by

$$\boldsymbol{A}^{-1} = (\boldsymbol{QR})^{-1} = \boldsymbol{R}^{-1}\boldsymbol{Q}^{-1} = \boldsymbol{R}^{-1}\boldsymbol{Q}^{\mathsf{H}}. \tag{3.4}$$

Obtaining $\boldsymbol{Q}^{\mathsf{H}}$ is trivial, whereas taking the inverse of the upper-triangular matrix $\boldsymbol{R}$, which has real-valued diagonal elements, is more involved and requires the employment of back-substitution. Algorithms 1 and 2 provide the pseudo code for the modified Gram-Schmidt QRD and back substitution, with complexities of $\mathcal{O}(K^3)$ and $\mathcal{O}(K^2)$, respectively [34, 35]. The resulting matrices ($\boldsymbol{Q}^{\mathsf{H}}$ and $\boldsymbol{R}^{-1}$) are then multiplied according to Equation 3.4 in order to obtain the inverse of $\boldsymbol{A}$.

3.1.5 EXTENDED QRD

The calculation of the QRD, as mentioned in the previous section, involves the extra step of obtaining the inverse of $\boldsymbol{R}$, which can be circumvented by recruiting the extended QRD algorithm [36]. Doing so involves the extension of matrix $\boldsymbol{A}$ with an identity matrix, which yields a $2K \times K$ matrix given by

$$\boldsymbol{A}_{\text{ext}} = \begin{bmatrix} \boldsymbol{A} \\ \boldsymbol{I}_K \end{bmatrix}. \tag{3.5}$$

Application of the QRD algorithm on this extended matrix leaves us with

$$\boldsymbol{A}_{\text{ext}} = \begin{bmatrix} \boldsymbol{A} \\ \boldsymbol{I}_K \end{bmatrix} = \boldsymbol{Q}_{\text{ext}}\boldsymbol{R}_{\text{ext}} = \begin{bmatrix} \boldsymbol{Q}_1 \\ \boldsymbol{Q}_2 \end{bmatrix} \boldsymbol{R}, \tag{3.6}$$

where

$$\boldsymbol{Q}_2\boldsymbol{R} = \boldsymbol{I}, \tag{3.7}$$

Algorithm 3 Extended QRD

Input: $K \times K$ complex matrix $\boldsymbol{A}$
Output: Unitary matrix $\boldsymbol{Q}$ and $\boldsymbol{R}_{inv}$

```
Q ← A
R_inv ← I_K
for i ← 1 to K do
    r1 ← ‖Q(:,i) + R_inv(:,i)‖_2
    Q(:,i) ← Q(:,i)/r1
    R_inv(:,i) ← R_inv(:,i)/r1
    for j ← i + 1 to K do
        r2 ← Q(:,i) · Q(:,j) + R_inv(:,i) · R_inv(:,j)
        Q(:,j) ← Q(:,j) − Q(:,i)r2
        R_inv(:,j) ← R_inv(:,j) − R_inv(:,i)r2
    end for
end for
```

which implies

$$\boldsymbol{R}^{-1} = \boldsymbol{Q}_2. \tag{3.8}$$

Now the derivations of $\boldsymbol{Q}$ and $\boldsymbol{R}^{-1}$ happen organically and the need for explicit back-substitution is obviated. Algorithm 3 lists the pseudo code for the extended QRD algorithm.

3.1.6 CHOLESKY

The Cholesky decomposition [33] factorizes the complex matrix $\boldsymbol{A}$ into an upper triangular matrix and its conjugate transpose, expressed mathematically as

$$\boldsymbol{A} = \boldsymbol{L}^{\mathsf{H}}\boldsymbol{L}. \tag{3.9}$$

The inverse of A is then calculated with

$$\boldsymbol{A}^{-1} = (\boldsymbol{L}^{\mathsf{H}}\boldsymbol{L})^{-1} = \boldsymbol{L}^{-1}(\boldsymbol{L}^{\mathsf{H}})^{-1}, \tag{3.10}$$

in which $\boldsymbol{L}$ is an upper triangular matrix, and $\boldsymbol{L}^{\mathsf{H}}$ is a lower triangular matrix. Consequently, $\boldsymbol{L}^{-1}$ and $(\boldsymbol{L}^{\mathsf{H}})^{-1}$ can be acquired using backward-substitution and forward-substitution, respectively. The complexity of the decomposition itself is $\mathcal{O}(K^3)$, while both of the substitutions have a complexity of $\mathcal{O}(K^2)$. The final inversion of the matrix $\boldsymbol{A}$ is then carried out by a simple matrix

Algorithm 4 Cholesky

Input: $K \times K$ complex matrix $\boldsymbol{A}$
Output: Lower triangular matrix $\boldsymbol{L}$ such that $\boldsymbol{L}\boldsymbol{L}^H = \boldsymbol{A}$
1: $\boldsymbol{L} \leftarrow \boldsymbol{0}_K$
2: $\boldsymbol{s} \leftarrow \boldsymbol{0}$
3: **for** $i \leftarrow 1$ **to** K **do**
4: $\quad \boldsymbol{s}(i:K) \leftarrow \boldsymbol{A}(i:K, i)$
5: $\quad$ **for** $j \leftarrow 1$ **to** $i-1$ **do**
6: $\quad\quad \boldsymbol{s}(i:K) \leftarrow \boldsymbol{s}(i:K) - \boldsymbol{L}(i:K, j) * l_{ij}$
7: $\quad$ **end for**
8: $\quad \boldsymbol{L}(i:K, i) \leftarrow \boldsymbol{s}(i:K) / \sqrt{\boldsymbol{s}(j)}$
9: **end for**

Algorithm 5 Forward substitution

Input: $K \times K$ upper-triangular matrix $\boldsymbol{A}$
Output: $\boldsymbol{B}$ as the solution to $\boldsymbol{A}\boldsymbol{B} = \boldsymbol{I}_K$
1: $\boldsymbol{B} \leftarrow \boldsymbol{I}_K$
2: **for** $i \leftarrow 1$ **to** K **do**
3: $\quad$ **for** $j \leftarrow 1$ **to** $i-1$ **do**
4: $\quad\quad \boldsymbol{b}_i \leftarrow \boldsymbol{b}_i - a_{ij}\boldsymbol{b}_j$
5: $\quad$ **end for**
6: $\quad \boldsymbol{b}_i \leftarrow \boldsymbol{b}_i / a_{ii}$
7: **end for**

multiplication according to Equation 3.10. The pseudo code for the Cholesky algorithm is listed in Algorithm 4, and Algorithm 5 does the same for forward substitution.

3.2 DISTRIBUTED MASSIVE MIMO

Let us now examine the algorithms from the DMIMO point of view. There are different DMIMO algorithms, e.g., [37] [38], but for the discussion here we make use of the linear detection algorithm with distributed Iterative Interference Cancellation (IIC) in [37] as an example. The purpose of processing distribution is to achieve local dimensionality reduction and local data consumption [39]. Assuming a panelized architecture, the p-th panel calculates its own $M_p \times N_p$ detection matrix $\mathbf{Q}_p$, which is then applied to the $M_p \times 1$ received vector $\mathbf{y}_p$ at the said panel according to

Algorithm 6 Decentralized IIC algorithm for the p-th panel.

Input: $K \times K$ covariance matrix $\boldsymbol{Z}_{p-1} from preceding neighboring panel$
Input: $M_p \times M_p$ local channel matrix $\boldsymbol{H}_p$
Output: $K \times K$ covariance matrix $\boldsymbol{Z}_p$ to succeeding neighboring panel
1: $[\boldsymbol{U}_p, \boldsymbol{S}_p] = \text{SVD}(\boldsymbol{Z}_{p-1})$
2: $\tilde{\boldsymbol{H}}_p = \boldsymbol{H}_p \boldsymbol{U}_p \boldsymbol{S}_p^{-1/2}$
3: $\boldsymbol{Q}_p = \text{SVD}(\tilde{\boldsymbol{H}}_p)(:, 1 : N_p)$
4: $\boldsymbol{Z}_p = \boldsymbol{Z}_{p-1} + \rho \boldsymbol{H}_p^H \boldsymbol{Q}_p \boldsymbol{Q}_p^H \boldsymbol{H}_p$

$$\hat{\mathbf{x}}_p = \mathbf{Q}_p^H \mathbf{y}_p. \tag{3.11}$$

To obtain the final detection result, the locally computed $\hat{\mathbf{x}}_p$ vectors from the various panels need to be aggregated. This aggregation of partial results, which is carried out in a root processing node, is equivalent to

$$\hat{\mathbf{x}} = \sum \mathbf{Q}_p^H \mathbf{y}_p = \sum \hat{\mathbf{x}}_p. \tag{3.12}$$

In most cases not all the UEs are served by each panel in this DMIMO setup, and as a result the inter-panel interconnection bandwidth can be reduced by making N_p to be smaller than M_p. Each channel update triggers the generation of the detection matrix $\mathbf{Q}_p$, which is a solution to a local optimization problem that maximizes the sum-rate capacity with a dimensionality reduction constraint. The calculation of $\mathbf{Q}_p$ requires the $M_p \times K$ local channel state information $\mathbf{H}_p$, and the $K \times K$ $\mathbf{Z}_{p-1}$ matrix received from the previous immediate neighboring panel. This latter matrix is a covariance matrix that holds information about existing inter-user-interference.

Algorithm 6 lists the steps that comprise the IIC method. The $\mathbf{Z}_{p-1}$ contribution matrix is first factorized using SVD ([37]), yielding

$$[\mathbf{U}_p, \mathbf{S}_p] = \text{SVD}(\mathbf{Z}_{p-1}), \tag{3.13}$$

in which $\mathbf{S}_p$ is the diagonal matrix housing the singular values, and $\mathbf{U}_p$ is the left unitary matrix. Now, dimensionality reduction is achieved by obtaining the $\mathbf{Q}_p$ matrix, which contains the N_p strongest singular vectors of $\tilde{\mathbf{H}}_p$, calculated by

$$\tilde{\mathbf{H}}_p = \mathbf{H}_p \mathbf{U}_p \mathbf{S}_p^{-1/2}, \tag{3.14}$$

$$\mathbf{Q}_p = \text{SVD}(\tilde{\mathbf{H}}_p)(:, 1 : N_p). \tag{3.15}$$

In the last step of local processing, the new covariance matrix $\mathbf{Z}_p$ is computed and will be passed down to the neighboring panel for interference mitigation purposes, according to

$$\mathbf{Z}_p = \mathbf{Z}_{p-1} + \mathbf{H}_p^H \mathbf{Q}_p \mathbf{Q}_p^H \mathbf{H}_p. \tag{3.16}$$

One thing to take away from the discussion above is that SVD is a potent weapon in a researcher's arsenal, and one could even say that it is the crown jewel of linear algebra. Hence, its efficient calculation is all too apparent, and I linger a bit here by taking a detour and devoting the next section to further look into this aspect.

3.2.1 SVD

The time-honored SVD is arguably the holy grail of matrix decompositions [40], and finds utility in modern science and engineering in a wide variety of systems, that run the gamut from wireless communications to artificial intelligence [37, 41–43].

The two main methods of SVD calculation are based on the Jacobi algorithm [44] and the Golub-Kahan (GK) algorithm [45, 46]. In the former, rotations are applied to the input matrix in sweeps to nullify the off-diagonal elements. These rotations are unitary transformations performed by left and/or right multiplications, either in two-sided or one-sided manner. The GK method, on the other hand, follows a two-phased methodology, with the first phase performing matrix bidiagonalization, and the second phase carrying out diagonalization of the bidiagonal input matrix. Both of these algorithms undergo repetitions in an iterative manner, which continues until the singular values converge.

Here, I will use the GK algorithm as described in [42], which as mentioned above takes a dual-phase approach to the problem. The first phase of decomposition yields three matrices, two of which are unitary matrices which flank a third bidiagonal matrix. Thereafter, the bidiagonal matrix undergoes rotation transformations (by application of Givens rotations [47]), which ends up with the desired diagonal singular values matrix.

The first phase of bidiagonalization can be performed in different ways. For instance, using Givens rotations to zero out one element at a time, or applying Householder reflections to introduce multiple zeros at once. Figure 3.3 depicts how the latter method is applied for a small 4×4 toy example matrix. Each step involves picking a partial column or row $\mathbf{x}$, and zeroing out all of its

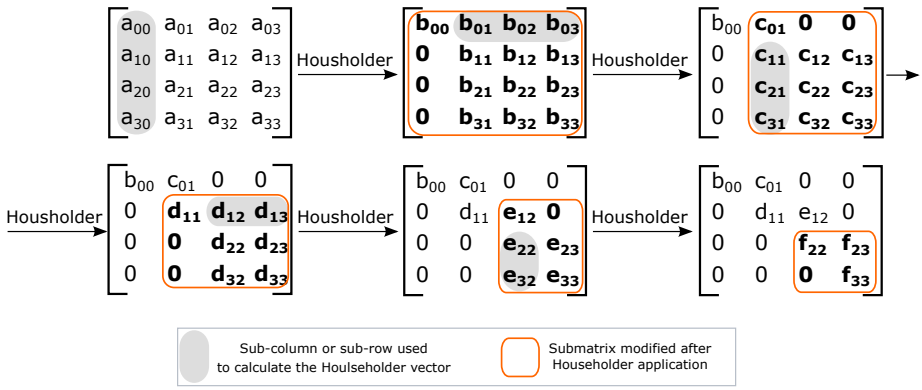

Figure 3.3: Example 4 × 4 matrix bidiagonalization using the Householder reflections.

elements but the first one. In order to calculate the Householder vector $\mathbf{v}$ the first element of vector $\mathbf{x}$ is updated, as specified by

$$x_1 + \frac{x_1}{\|x_1\|}\|\mathbf{x}\|, \tag{3.17}$$

and this updated $\mathbf{x}$ is used to obtain the Householder vector by

$$\mathbf{v} = \frac{\mathbf{x}}{\|\mathbf{x}\|}. \tag{3.18}$$

The submatrix $\mathbf{B}_{i:j:} = \mathbf{B}(i\ :; j\ :)$ of the input matrix $\mathbf{B}$ is then updated according to

$$\mathbf{B}_{i:j:} - 2\mathbf{v}\mathbf{v}^*\mathbf{B}_{i:j:}, \tag{3.19}$$

$$\mathbf{B}_{i:j:} - 2\mathbf{B}_{i:j:}\mathbf{v}\mathbf{v}^*, \tag{3.20}$$

for the column and row cases, respectively. After the descent down the diagonal is complete, we are left with a bidiagonal matrix. The undesirable non-zero elements on the superdiagonal are chased down along the diagonal by successively applying Givens rotations. This nullification process is visualized in Figure 3.4, which shows one iteration that results in a new bidiagonal matrix with superdiagonal elements that have diminished values. This constitutes one iteration, and is repeated until the superdiagonal elements are nullified, and the diagonal values converge to their singular real-valued cousins. The SVD calculation is not trivial, and its computational complexity for an input $N \times N$ complex-valued matrix $\mathbf{A}$ is proportional to $\mathcal{O}(N^3)$.

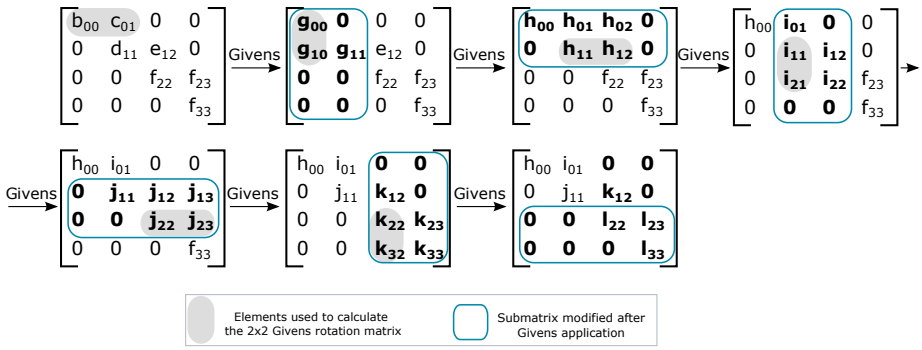

Figure 3.4: Example 4×4 matrix diagonalization step using Givens rotations.

3.3 KERNEL OPERATIONS AND REQUIREMENTS

The algorithms listed in the previous sections can act as guidelines as to what operations are needed in order to support efficient processing. Table 3.1 collects a list of kernel operations that are constituent parts of the mentioned algorithms.

By inspecting these kernel operations we can derive these guidelines:

- The system needs to be able to handle basic vector operations efficiently. This includes operating on multiple elements in parallel (e.g. vector-vector addition).
- Vector reduction operations (e.g. vector norm) appear frequently.
- Scalar-vector operations are also typical (e.g. scalar-vector multiplication).
- Vector element access is commonly needed, either to modify an element or to extract it for use in another operation.
- Higher-order matrix operations (e.g. matrix-matrix multiplication) benefit from special treatment, beyond what vector operations can provide on their own.
- Some calculations (e.g. SVD) can take advantage of a dedicated unit specially customized for the task.

These guidelines will help us in our decision-making abilities when it comes to the micro-architectural development covered in the next chapter.

To help analyze the performance requirements, let us take a look at the ZF detection formula again, as per

$$(\boldsymbol{H}^{\mathsf{H}}\boldsymbol{H})^{-1}\boldsymbol{H}^{\mathsf{H}}\boldsymbol{r}. \tag{3.21}$$

Table 3.1: Example kernel operations.

Description	Symbol	Operation
Matrix multiplication	$\boldsymbol{A} \times \boldsymbol{B}$	$\boldsymbol{C}_{ij} = \sum_k \boldsymbol{A}_{ik} \times \boldsymbol{B}_{kj}$
Hadamard (element-wise) vector product	$\boldsymbol{a} \odot \boldsymbol{b}$	$(\boldsymbol{a})_i \times (\boldsymbol{b})_i$
Element-wise vector addition/subtraction	$\boldsymbol{a} \pm \boldsymbol{b}$	$(\boldsymbol{a})_i \pm (\boldsymbol{b})_i$
Vector dot product	$\boldsymbol{a} \cdot \boldsymbol{b}$	$\sum_i (\boldsymbol{a}_i \times \boldsymbol{b}_i^*)$
Vector-scalar product	$\boldsymbol{a} \times s$	$(\boldsymbol{a})_i \times s$
Vector element modify	$\boldsymbol{a}[n] = s$	$\boldsymbol{a}_n = s$
Vector-vector element product	$\boldsymbol{a} \times \boldsymbol{b}[n]$	$(\boldsymbol{a})_i \times \boldsymbol{b}_n$
Vector norm	$\|\boldsymbol{a}\|$	$\sqrt{\boldsymbol{a} \cdot \boldsymbol{a}}$
Vector norm inverse	$1/\|\boldsymbol{a}\|$	$1/\sqrt{\boldsymbol{a} \cdot \boldsymbol{a}}$

This needs a Gramian matrix calculation ($\boldsymbol{H}^{\mathsf{H}}\boldsymbol{H}$), Gramian inversion, one matrix-vector multiplication ($\boldsymbol{H}^{\mathsf{H}}\boldsymbol{r}$), and one final matrix-vector multiplication. I showed in the previous chapter that even a system with 20 MHz bandwidth requires 210k channel inversions per second. This amounts to an allotted time of just below 3 µs per inversion! For a more complete analysis let us compile the numbers for different user speeds. To calculate the coherence times for different UE speeds [35] we can use

$$T_p = \frac{0.69c}{2\pi v_{max} f_c}, \tag{3.22}$$

where T_p is the time between uplink pilots, v_{max} the maximum supported UE speed, c the light speed, and f_c the carrier frequency. Assuming a carrier frequency of $f_c = 3.7$ GHz is selected, Table 3.2 lists the requirements for two different systems, one with 20 MHz bandwidth and the other with 50 MHz bandwidth, both with a subcarrier spacing of 15 kHz. This further substantiates the significance of efficient matrix processing capabilities.

Table 3.2: Requirements in terms of inversions/s for two different systems.

UE Speed [km/h]	5	50	100
Coherence Bandwidth n_{b} [#subcarriers]	16	16	16
Coherence Time [ms]	7.2	0.73	0.36
Coherence Time n_{t} [#OFDM symbols]	100	10	5
#Inversions/s (20 MHz bandwidth)	10.5 k	105 k	210 k
#Inversions/s (50 MHz bandwidth)	26 k	262 k	525 k

Table 3.3: Information rate requirements for example scenarios.

Subcarrier spacing	Bandwidth	Total subcarriers	#OFDM symbols in one second	#Resource elements in one second	#Users	#Modulation scheme	Information rate (bps)
15 kHz	20 MHz	1200	14000	16.8M	8	32-QAM	672M
15 kHz	20 MHz	1200	14000	16.8M	8	64-QAM	806M
15 kHz	50 MHz	3300	14000	46.2M	8	64-QAM	2217M
30 kHz	100 MHz	3300	28000	92.4M	8	64-QAM	4435M
60 kHz	100 MHz	1650	56000	92.4M	8	64-QAM	4435M
60 kHz	100 MHz	1650	56000	92.4M	16	64-QAM	8870M
60 kHz	200 MHz	3300	56000	184.8M	16	128-QAM	20700M

When it comes to the data rate requirements, we can use the following

$$I = N_r K M_{bits}, \tag{3.23}$$

where I is the information rate, N_r the total number of resource elements contained in one second, K number of users, and M_{bits} the number of bits used for modulation. Table 3.3 details the information rate for a number of different scenarios in tabular form[2]. So, for instance, for a 20 MHz-bandwidth system using a subcarrier spacing of 15 kHz, with $K = 8$ users, and 64-Quadrature Amplitude Modulation (QAM) scheme the information rate I will come down to 806 Mbps. Cranking up the bandwidth to 100 MHz and subcarrier spacing to 60 kHz, and leaving the other factors intact, will change the rate to 4,435 Mbps. It is already evident form this investigation that the hardware must not only be pliant in dealing with the varied number of algorithms, but it also needs to be performant. This is a subject that merits its own chapter, and that is the topic that I will flesh out in the one that follows.

[2]The numbers in the table are chiefly to show the scaling. In reality, the modulation scheme in some of the cases is probably of a lower order.

4

Processing Architecture and Digital Hardware

DIGITAL systems have transformed the way we approach life in a dramatic fashion. The pace at which more and more[1] components could be put into a chip was so consistent that it was given a law, namely the Moore's Law [48, 49]. This fast growth and the accompanying complexity meant that designs took different paths and fell into certain categories over time. In this chapter, I will take a brief look into each one, and then select a combination in order to implement the communication and positioning efficiently. Before engaging the downward descent into various architectural techniques, it has to be noted that without proper software support even the best hardware design falls short[2].

4.1 DIFFERENT APPROACHES

In the discussion that follows, I will sift through some of the common categories that are used to classify hardware architectures according to certain criteria. Figure 4.1 helps to visualize roughly where in the performance-flexibility spectrum each of these architectures fall into. But the caveat is that not every design falls squarely into just one category, and the line between these categories is blurred, in a way that makes shoehorning a certain design into one class nigh impossible. Nevertheless, this catalog, amorphous as it may seem sometimes, assists us in wading through the complex process of picking the right architecture for the job at hand.

[1] One could say: Moore and Moore.

[2] As Michael Kagan, NVIDIA's CTO elegantly put it: "Without software, chips are just expensive sand" [50].

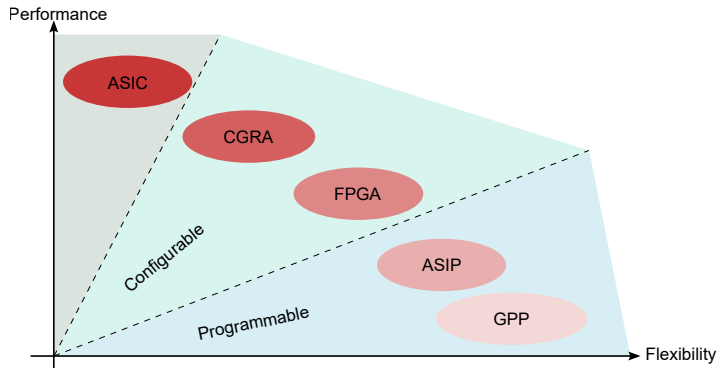

Figure 4.1: Flexibility vs. performance.

4.1.1 ASIC

An ASIC is a chip that is fully, or partially, customized for a specific application, or set of applications. Full-custom ASICs are built from the ground up, whereas semi-custom ASICs can take advantage of pre-designed functional blocks, in order to reduce design complexity. Full customization means that the designers can even go down to the transistor level. The designers choose the ASIC route if the design specifications put a premium on area and/or energy efficiency, low latency, high throughput, or a combination of the above.

There exist many ASIC designs that are made to cater to specific areas. The Tensor Processing Unit (TPU) by Google [51,52] is a custom ASIC with tens of thousands of small Multiply ACcumulate (MAC) units and a large, software-managed, on-chip memory, specially tuned for accelerating the inference phase of Neural Networks (NNs). Since their introduction, there have been various versions of TPUs with focus on training and inference [53, 54]. FFT and its inverse are central to many applications, and low-latency requirements all but make an ASIC implementation mandatory [55]. The decoding process in MIMO detection is yet another case where a high-throughput ASIC implementation is a must [56]. Other example ASICs that cover a wide spectrum of fields include [56,57].

4.1.2 CONFIGURABLE ARCHITECTURES

As tantalizing as it sounds to utilize ASICs for every design, in practical terms it is not always feasible, or sometimes even desirable, due to manifold reasons, inducing the exorbitant costs of designing ASICs, the required knowhow, and the time-to-market constraints. One middle-ground approach to achieve acceptable performance while keeping costs low is to use configurable archi-

tectures. In these systems the logic, routing, or both are configurable. The amount of configurability depends on the level of granularity. In systems like Field-Programmable Gate Arrays (FPGAs) the granularity is low and down to the bit level, while on the opposite side of the isle sit Coarse-Grained Reconfigurable Architectures (CGRAs), where huge logic blocks can be configured to select among a handful of operations. The three prominent characteristics of CGRAs can be summarized as, domain specific flexibly, spatial and temporal configurability, and data-driven execution [58].

One could argue that the inception of configurable architectures was materialized all the way back in the Electronic Numerical Integrator and Computer (ENIAC) machine [59], which was configured by program switches and plug-board wirings. Xilinx started the highly-successful FPGA computing era [60] when they introduced their first product[3] [61] in 1985. There are numerous examples of CGRAs introduced over the years such as MATRIX [62], in which instruction storage and distribution and data storage and computation resources are united, Plasticine [63], which is a CGRA with direct support for parallel patterns, SARA [64], a reconfigurable dataflow accelerator with distributed memory and more flexible and large-scale datapaths, and DSAGEN [65], that utilizes a small number of hardware primitives and a primitive-aware compiler that almost achieves the performance of hand-tuned programs. Other notable systems can be found in [66–69].

4.1.3 PROGRAMMABLE SYSTEMS

GENERAL-PURPOSE CPU

General-purpose Central Processing Units (CPUs) are a class of digital computing systems that are designed to, in crude terms, handle anything you can throw at them[4]. These architectures run by executing commands issued based on a finite set of instructions[5]. These instructions can be either very simple or highly complex, and the proponents of each went on to create two schools of thought. If the whole Instruction Set Architecture (ISA) is designed based on simple instructions, the architecture is considered to follow the Reduced Instruction Set Computer (RISC) principles, and if the machine is composed of complex instructions mixed in with the simple ones, it is known as Complex Instruction Set Computer (CISC).

The first RISC machine can be traced back to CDC 6600 [70], designed way back in 1964, followed by the IBM 801 [71] as the first modern machine that

[3] Xilinx XC2064, the first commercially viable FPGA.

[4] Excluding the kitchen sink.

[5] So, if the processor could speak, the instructions would be its words and the whole set of instructions its vocabulary.

ushered in the RISC era, while the CISC machines are as old as Methuselah[6] himself, and seem to have been present since the Big Bang! The most promising RISC architecture today is the RISC-V[7] ISA [72, 73], which is an open ISA[8]. The battle between the two methodologies raged on for many years, but due, partly, to the seminal works such as [74], and, mostly, to their inherent simplicity, the RISC systems proved too much for their CISC counterparts in the end, and the industry finally settled on the former. This by no means indicates that CISC is dead; on the contrary, CISC designs are still alive and kicking. Case in point, the most popular ISA for PCs, workstations and cloud computing as of 2024 is based on the CISC x86 architecture, with a software developer's manual spanning over 5000 pages! [75, 76]. The catch is that these systems internally translate the complex instructions to their simpler cousins on the fly, and that is how they coexist peacefully with pure RISC machines, at least for now.

This category has a wide variety of architectures, but its principal selling point is its ability to perform any functionality asked by the user according to a program, written mostly in high-level languages nowadays. This extreme flexibility comes at the cost of losing performance, to the the point that General Purpose Processors (GPPs) pale in comparison to the custom designs when it comes to performance in a specific task. But this did not stop researchers and computer architects to try and find ways to amp up performance through clever techniques such as pipelining, branch prediction, and exploiting memory hierarchies. Despite these valiant efforts, the domain-neutral CPU does not hold a candle to architectures that are more conversant with parallelization of large amounts of compute, as I will show next.

GPU

A Graphics Processing Unit (GPU) is a special from of a CPU that really shines in applications made specifically to run graphics. But over time these processors have evolved and are now capable of taking on much more than just graphics. One distinguishing difference to CPUs is the size of each processing unit and the number of processing units available on the chip. If CPUs feature from a few to about a 100 beefy cores, the GPUs can boast core counts in the thousands[9], albeit cores that are skinny by comparison. With

[6]Noah's grandfather in the Bible, a patriarchal figure purported to have lived to the ripe age of 969, the longest ever claimed!

[7]Pronounced "risk-five".

[8]Sometimes referred to as an open-source ISA, which is a misnomer, as there is no source code here. It is just an open architecture specification.

[9]For instance, NVIDIA's top-of-the-line GeForce RTX 4090 GPU packs quite a punch with a whopping 16,384 cores [77].

these many processing cores the GPUs can launch thousands of threads of execution in a so-called Single Instruction Multiple Threads (SIMT) execution model. SIMT is a close cousin of the SIMD model, where each instruction is applied to multiple data items at the same time to gain performance at the cost of area. The many threads endow GPUs with tremendous parallel processing capabilities, and render[10] them a treasure trove of compute power to tap into for embarrassingly parallel workloads. Therefore, it is no surprise that the standard-bearer for high-performance computing appearing in all the top 10 supercomputers in the world is the GPU [78].

ASIP

The final puzzle piece in the programmable processor category is the ASIP. An ASIP is similar in concept to a general-purpose CPU, with the exception that it is customized to run certain applications much more efficiently. What this means is that the processor's ISA is extended/modified to include instructions that are usually omitted in a general-purpose architecture, for instance instructions that are targeted at, and expedite, communications algorithms, audio/video processing, and machine learning, to name a few. So even though ASIPs might be able to run any application, that is not their life's purpose [11].

Additionally, the software tools that enable designing ASIPs play an important role. Not only do they make designing ASIPs easier, but they also provide the compiler for the designed processor in a process known as compiler-in-the-loop. This shortens the design time and lowers complexity, as the designers can focus their efforts on the hardware, while the tool relieves the hardware designer from the tedium of creating a compatible compiler. Figure 4.2 demonstrates the compiler-in-the-loop concept. The designer provides a description of the processor's ISA and microarchitecture, detailing the available instructions, pipeline structure and functional units in a dedicated language such as nML [79]. The ASIP Electronic Design Automation (EDA) tool then generates a compatible compiler tool chain, including the assembler, linker, debugger, profiler, and instruction-set simulator. The designer then can use these tools to check correct functionality and desired performance, and if not satisfied go back and change the processor model, and rinse and repeat. Finally, when the design meets the criteria, the ASIP tool's Register Transfer Level (RTL) generator can be invoked to obtain the processor's description in

[10] Pun intended ☺.

[11] When it comes to an ASIP's worldview, and to take a page from George Orwell's Animal Farm: all instructions are equal but some instructions are more equal than others.

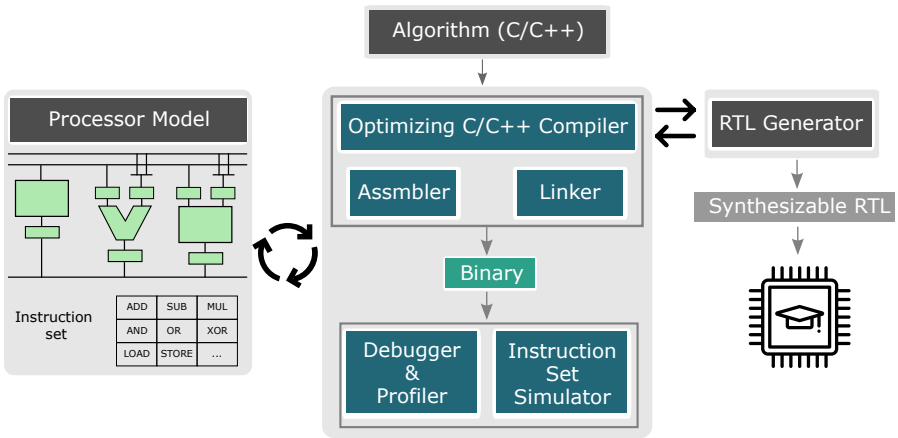

Figure 4.2: ASIP flow.

a Hardware Description Language (HDL). The design can then be synthesized and implemented on an FPGA or taped out to be manufactured into a chip.

On the surface it might seem that ASIPs are a new concept, but their origin can be traced back to the 1980s. The White Dwarf [80] is one of the first of such application-specific processors that came with a retargetable compiler able to generate highly parallel and efficient code for the processor. The work in [81] describes the hardware-software co-design in an ASIP setting, and some more recent examples can be found in [82, 83]. The notable tools by major silicon companies are ASIP Designer [84] by Synopsys, Tensilica Xtensa [85] from Cadence, and Codasip Studio [86] by Codasip. For the proprietary-averse user, OpenASIP [87, 88] provides an open-source alternative.

4.1.4 HYBRID ARCHITECTURE

Systems nowadays are highly complex, and in order to meet their computational needs seek help from specialized computing resources. Even though ASIC architectures fill a niche that hit the efficiency jackpot, sacrificing some performance in favor of gaining flexibility is more crucial in certain situations. For instance, in the communications sphere this applies to BSs that need to support multiple wireless technologies, e.g. LTE, LTE-A, 5G and beyond. Consequently, a heterogeneous architecture, with a combination of ASIP to allow programmability and specialized units to attain performance goals, hits the sweet spot between flexibility and efficiency, and that is the route I have taken for this work, which will be expounded upon in the upcoming sections.

4.2 DESIGN CONSIDERATIONS

The following subsections will touch on some of the factors that help steer us towards a suitable design.

4.2.1 PROGRAMMABILITY

Did I mention that we are in the golden age of computer architecture with accelerators paving the way? Yes, I did [7]. But in this volatile field, fixed-function implementations alone can not cope with the ever-changing requirements and could become obsolete quickly. I showed in chapter 3 that there are a variety of algorithms and computational methods that need to be supported. As a result, flexibility is highly prized [89], and algorithm-hardware co-design methodology provides a satisfactory answer, with ASIPs as a convenient vehicle to facilitate this process [90]. The resulting hardware is flexible enough to run different programs at a high abstraction level (e.g. C language), and adapt to the vagaries of the field, and at the same time it can benefit from fixed-function units by integrating it seamlessly with the rest of the processor. Therefore, a RISC architecture with some CISC enhancements fits the bill perfectly.

4.2.2 ACCELERATION AND DATA-LEVEL PARALLELISM

Even though flexibility is an important factor in allowing the design to adapt itself to disparate circumstances, there are tasks that lend themselves to customized architectures that lean towards more hardwired functionality. As I demonstrated in the preceding chapters, matrix computations feature prominently in the algorithms encountered in communications and positioning applications. Giving some of the key operations such as GEMM, SVD, and CNN special treatment, by executing them on dedicated customized units, will be highly beneficial. As a result, the processor can gear itself with a systolic array to expedite GEMM, an SVD accelerator to tackle the SVD factorization, and a CNN module to cater to the demands of CNN computations. All of these units compensate the loss in programmability with commensurate gain in performance, and on top of that, they are kept visible to the programmer through compiler intrinsics[12] that allow easy access in code and the chance to configure the said units.

Many of the algorithms discussed up to this point show inherent data-level parallelism. The SIMD execution model is a good fit to exploit this parallelism,

[12]Intrinsics are hardware-aware functions in a high-level programming language that allow developers to directly access low-level machine instructions or processor features.

and can be made to work very well inside a processor pipeline. Exposing the SIMD parallelism in the programming model[13] of the ASIP platform is pretty straightforward. In this model the overhead of instruction fetch and dispatch is amortized over multiple data items. Due to its versatility, I have used it in the implementation of the ASIP itself as well as the accelerators attached to the processor. The sub-sections under the microarchitecture section will take a peek at the details of the implementation.

4.2.3 MEMORY ACCESS PATTERNS

Manipulating a matrix requires access to its elements, which sometimes manifests itself as access to multiple elements at the same time, most notably in the form of rows, columns and diagonals. Now, when it comes to hardware, every access has a cost, and having fast and uniform access to any row, column, or diagonal of a matrix is not only desirable, but essential.

In order to attain a better grasp of the impact of the different operations required in massive MIMO baseband processing on access patterns. These operations, along with the required access patterns, are listed in Table 4.1. Keeping this in mind, the designed hardware must be able to aggregate access to multiple elements in as few cycles as possible. In order to realize this, a dedicated memory unit is developed that, in cooperation with the compiler, brings visibility to the programmer, as discussed in the microarchitecture section.

Table 4.1: Different operations in massive MIMO processing and the needed access modes.

Operation	Access modes
$\boldsymbol{H}^H\boldsymbol{H}$	Row
$\boldsymbol{H}^H\boldsymbol{H} + \alpha\boldsymbol{I}$	Row, Diagonal
$(\boldsymbol{H}^H\boldsymbol{H} + \alpha\boldsymbol{I})^{-1}$	Row, Diagonal, Column
General matrix multiplication	Row or Column
SVD	Row, column and diagonal

4.3 MICROARCHITECTURE

Now it is time to put the points that I discussed above into practice. I will start with the ASIP itself as the focal architectural point, with the rest of the

[13]Refer to chapter one.

Table 4.2: Design considerations and requirement.

Tasks	**Flexibility**	**Parallelism**	**Memory access efficiency**
Detection	ZF, MMSE, MR Cholesky, QRD, ...	SIMD, partial SIMD, element-wise	Medium
GEMM	Matrix size (#antennas, #users)	SIMD	High
2D FFT	Matrix size (#antennas, #subcarriers)	SIMD	High
SVD	Matrix size	SIMD, partial SIMD, element-wise	Medium
CNN	Layer structure	Mostly SIMD	High

units revolving around it. For this I am going to use the processors presented in papers IV and V as examples. Table 4.2 maps the tasks that need to be carried out to the different design considerations and requirements. Armed with this information, the detection process, due to its higher flexibility requirements and less efficient access patterns, can avail itself of the more malleable ASIP, while the other tasks can subscribe to configurable or fixed-function accelerators, in order to bring the full power of the processor to bear. With this analysis in mind, I will put on my hardware designer hat, and, in what follows, look at the proposed architectural details.

4.3.1 THE ASIP

Figure 4.3 diagrams the bird's-eye view of the processor and the accompanying accelerators and memories, with the supportive connective tissue. The ASIP was designed with the ASIP Designer [84] tool from Synopsys, and is based on the up-and-coming RISC-V architecture. But the baseline RISC-V is not up to par with the computational demands of the algorithms we have seen so far, and as a result, the processor comes equipped with a 16-wide SIMD vector core. This core has its own vector register file and execution pipeline, and has enough accessories to make it able to support the kernel operations that I tabulated in Table 3.1.

This vector core amplifies the processing capabilities of the processor, but it falls short to satiate all the computational demands. Thankfully, the more customized units come to the rescue, in the form of a 16×16 systolic array and an SVD accelerator, that significantly boost the processor's matrix-related

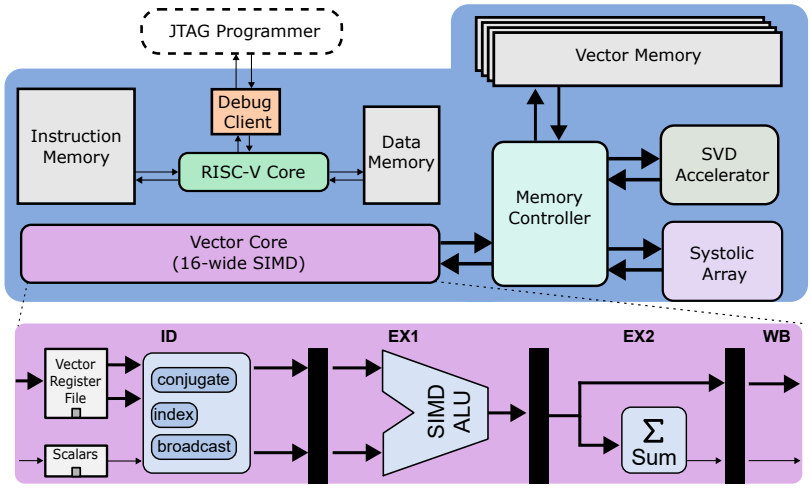

Figure 4.3: 1000-foot view of the ASIP.

performance. Finally, all of these additions to the processor would be utterly useless if the memory subsystem was not able to follow suit. To remedy this, a memory controller and parallel memory unit cooperate with the processor and the accelerators to facilitate fast access to matrix elements.

4.3.2 MULTI-ACCESS-MODE PARALLEL VECTOR MEMORY

I already established that we want swift access to multiple matrix elements to efficiently perform calculations. To make this a reality, a multi-access-mode parallel vector memory [91] is provided in hardware that enables one-cycle access to a row/column/main diagonal of a matrix. This is accomplished by utilizing many banks in the vector memory, which are straddled by data shufflers, as sketched out in Figure 4.4. The memory controller takes the role of an arbiter that mediates between the different units that vie for access to the memory. The input data shuffler swizzles the incoming vector elements based on the requested access mode (row/column/diagonal) for memory writes, and the output shuffler does the same thing for memory reads. In case of 16 memory banks this setup enables one-cycle access to a row, column or the main diagonal of a 16×16 matrix. For bigger matrices an extra cycle is added for each 16-element packet.

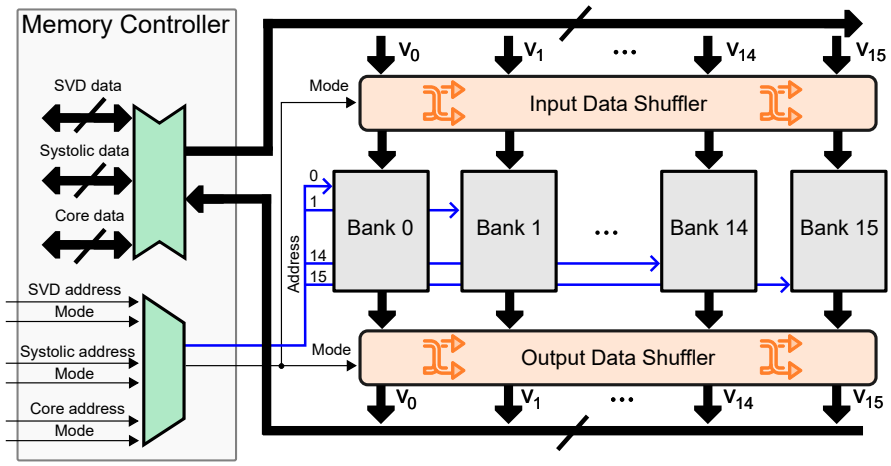

Figure 4.4: Parallel vector memory subsystem.

4.3.3 ACCELERATORS

As mentioned previously, the accelerators can aid the processor to take on heavier workloads. The SVD accelerator [42] for instance, is designed to speed up the SVD factorization and supports 8×8 and 16×16 complex-valued single-precision floating-point matrices. The large dynamic range provided by the floating-point representation makes the system more malleable when it comes to precision requirements. The data-level parallelism inherent in the two-phase GK algorithm dovetails nicely with the SIMD model, and hence provides a fertile ground for exploitation. This is manifested in the first phase by bi-diagonalizing the input matrix using Householder transformations, which is subsequently followed by a diagonalization phase involving the iterative application of Givens rotations until the singular values converge.

The affinity of the GK algorithm to vector arithmetic makes it a suitable candidate for SIMD implementation. Figure 4.5 captures the innards of the accelerator that adjoins the ASIP to keep a rein on the complexity of the SVD algorithm. In order to cache[14] the currently worked-on values, the accelerator has its own vector register and a scalar register buffers. Both phases of the GK algorithm are choreographed by a scheduler, which is a Finite State Machine (FSM) that is cajoled into action by the programmer in C code. The vector and

[14]Cache, also known as "little brain attic" as Sherlock Holmes put it: "A man should keep his little brain attic stocked with all the furniture that he is likely to use, and the rest he can put away in the lumber-room of his library, where he can get it if he wants it" [92].

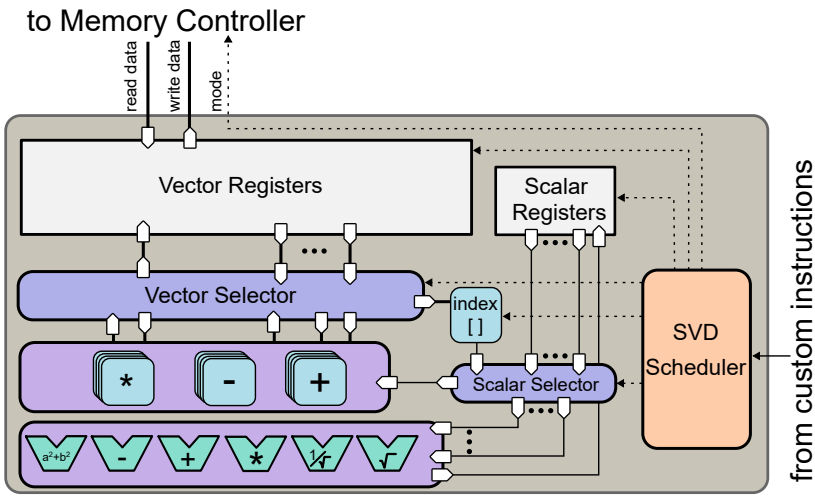

Figure 4.5: SVD accelerator.

scalar components comprising the datapath are the engines that support the required atomic operations, such as the square root and its inverse. The FSM dictates the flow of data between the different modules by picking the relevant data from the buffers, and steering them toward the appropriate datapath components. The SVD accelerator has a direct line of contact with the vector memory to make sure it does not starve for data.

The systolic array serves as another example of an accelerator that can tackle GEMM in a much more performant manner. The enlisted 16×16 array has a total of 256 Processing Elements (PEs) or cells[15], fully embracing the SIMD model of execution. Each PE is furnished with a MAC and internal storage, and the data marshaling is controlled in hardware by a programmer-facing scheduler. Custom instructions are used to get the accelerator off the ground running, by supplying matrix addresses and multiplication type to the scheduler, which then takes over to orchestrate the dataflow fully in-silicon.

The accelerators covered here are but mere examples. The ASIP can cooperate with a menagerie of accelerators as the need arises. For instance, the idea of using the CSI for positioning purposes has been explored before [93, 94]. The work in [95] presents a CNN module that uses CSI-based positioning and paper V incorporates that module in a mixed-datatype system with multiple vector memories.

[15]The term "cell" was coined by Robert Hooke since the organism reminded him of the monks' living quarters. In our scenario they look anything but.

Table 4.3: Throughput for the developed ASIP, for a system with the dimension of 128×16, utilizing the ZF algorithm and OFDM, running at 800 MHz.

System dimension	128×16		
UE speed [km/h]	5	50	100
System Parameters			
Coherence bandwidth n_b [#subcarriers]	16	16	16
Coherence time [ms]	7.2	0.73	0.36
Coherence time n_t [#OFDM symbols]	100	10	5
ASIP Performance			
Clock cycles to detect $n_b n_t$	39.7k	5.8k	3.9k
Calculation time to detect $n_b n_t$ [µs]	49.6	7.2	4.9
Throughput [Gb/s]	3.1	2.1	1.5

4.4 REQUIREMENTS AND PERFORMANCE EVALUATION

Section 3.3 in chapter 4 outlined the requirements from a system standpoint. Here I supply what the ASIP can do in terms of throughput. Table 4.3 shows[16] the ASIP capabilities in terms of detection throughput for a massive MIMO system with 128 antennas serving 16 UEs for ZF. The ASIP has the raw processing power of churning out 320k inversions/s, and by juxtaposing this number with those of the system requirements in Table 3.2, we can see that one ASIP can readily handle the task for a system operating in a 20 MHz bandwidth. For a much wider bandwidth of 100 MHz, multiple ASIPs need to be employed to be up to the mark.

[16]For a more complete analysis look up paper V.

5

Conclusion and the Path Ahead

In this thesis I made a case for using ASIPs as the bedrocks on which future digital wireless processing systems are built upon. The guidelines for DSA design, as mentioned in the outset in chapter 1, championed the use of dedicated memories, emphasized compute over fancy architectural constructs, asked for exposing the parallelism to the programming model, encouraged the use of customized data types, and espoused the use of domain-specific languages to ease the burden of porting the application. This set of guidelines is in line with the algorithm-hardware codesign methodology, which is perhaps the only viable way left to computer architects to squeeze every drop of performance out of the machines they design.

In chapters 2 and 3 I introduced the principles on which communications systems rely on and I listed the applications of future wireless systems and their concomitant requirements. The algorithms that make these systems tick were given central stage, and their constituent parts were laid bare. I then went a few rungs further down the ladder to break apart the algorithms into kernel operations that any system worth its salt must support. Finally, I capped things off by listing the system requirements.

I demonstrated that the ASIP flow lifts the hassle of compiler design and maintenance, while bringing the much-vaunted programmability to the table. This flexibility is crucial, as the algorithms and computations that need to be supported in current and future wireless systems cover a broad spectrum. But I also brought up the fact that modern designs rarely fall into a distinct category, and ASIPs are no exception to that rule. Consequently, heavier computational tasks can be siphoned off to accelerators as offload engines that can sit amicably alongside their ASIP guardians. In light of this, in chapter

4 I peered beneath the microarchitecture of an example ASIP paired with accelerators and custom memory subsystem to drive the point home.

5.1 THE PATH AHEAD

So, in this thesis I poked and prodded at the ASIPs and tried to convince the reader that, allied with accelerators, they earn their keep as vessels to navigate through the maze of algorithms. That said, there is a lot of work left to be done, and here I enumerate some of the most pressing ones that I think warrant some extra discussion.

- The ML models are getting bigger and more varied as time goes by. The CNN accelerator presented in **Paper I** is not able to cater to this wide range of models in its current incarnation. Therefore, I think it might be beneficial to trade off some performance in favor of flexibility by creating a tighter interaction between the ASIP and the accelerator. This can perhaps be achieved by paring down the accelerator into parallel primitive units, and deciding in code how those units are engaged.
- The SVD accelerator proposed in **Paper III** currently supports only 32-bit floating-point numbers. It would be interesting to try out reduced-precision flavors and measure their impact. Furthermore, the accelerator is currently optimized for matrix sizes equal to or smaller than 16×16. It could stand to benefit from a redesign to make it able to support bigger matrix sizes.
- The designs that I covered in this thesis concern themselves with positioning and communications scenarios. Another application that warrants attention is sensing, which is going to be crucial in automated systems and monitoring settings.
- One avenue of research could be the investigation of multiple-core ASIP designs, with heterogeneous cores featuring disparate accelerators and compute capabilities. This would be a much more challenging endeavor, but tools such as ASIP Designer can prove to be useful allies in bringing the problem to heel.

Bibliography

[1] I. Ghotbi, "Reconfigurable Receiver Front-Ends for Advanced Telecommunication Technologies," Doctoral Thesis (compilation). ISBN 978-91-8039-861-9 Oct. 2023.

[2] J. Bolhuis, I. Tattersall, N. Chomsky, and R. Berwick, "How Could Language Have Evolved?" *PLoS Biology*, 2014. doi: 10.1371/journal.pbio.1001934

[3] C. E. Shannon, "A mathematical theory of communication," *The Bell System Technical Journal*, vol. 27, no. 3, pp. 379–423, 1948. doi: 10.1002/j.1538-7305.1948.tb01338.x

[4] 3GPP, "Base Station (BS) radio transmission and reception (3GPP TS 38.104 version 18.6.0 Release 18)," 3GPP, Tech. Rep. ETSI TS 138 104 V18.6.0 (2024-08), 2024.

[5] E. Dahlman, S. Parkvall, and J. Skold, *5G NR: The Next Generation Wireless Access Technology*, 1st ed. USA: Academic Press, Inc., 2018. ISBN 0128143231

[6] H. Holma, H. Viswanathan, and P. Mogensen. (2021) Extreme massive MIMO for macro cell capacity boost in 5G-Advanced and 6G. [Online]. Available: https://onestore.nokia.com/asset/210786

[7] J. L. Hennessy and D. A. Patterson, "A new golden age for computer architecture," *Commun. ACM*, vol. 62, no. 2, pp. 48–60, jan 2019. doi: 10.1145/3282307. [Online]. Available: https://doi.org/10.1145/3282307

[8] J. L. Hennessy and D. A. Patterson, *Computer Architecture, Sixth Edition: A Quantitative Approach*, 6th ed. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2017. ISBN 0128119055

[9] M. Pellauer, Y. S. Shao, J. Clemons, N. Crago, K. Hegde, R. Venkatesan, S. W. Keckler, C. W. Fletcher, and J. Emer, "Buffets: An efficient and composable storage idiom for explicit decoupled data orchestration," in *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*, ser. ASPLOS '19. New York, NY, USA: Association for Computing Machinery, 2019. doi: 10.1145/3297858.3304025. ISBN 9781450362405 pp. 137–151. [Online]. Available: https://doi.org/10.1145/3297858.3304025

[10] "The bfloat16 numerical format." [Online]. Available: https://cloud.google.com/tpu/docs/bfloat16

[11] J. Ragan-Kelley, C. Barnes, A. Adams, S. Paris, F. Durand, and S. Amarasinghe, "Halide: a language and compiler for optimizing parallelism, locality, and recomputation in image processing pipelines," *SIGPLAN Not.*, vol. 48, no. 6, pp. 519–530, jun 2013. doi: 10.1145/2499370.2462176. [Online]. Available: https://doi.org/10.1145/2499370.2462176

[12] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin *et al.*, "TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems," 2015, software available from tensorflow.org. [Online]. Available: https://www.tensorflow.org/

[13] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. van den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot *et al.*, "Mastering the Game of Go with Deep Neural Networks and Tree Search," *Nature*, vol. 529, no. 7587, pp. 484–489, Jan. 2016. doi: 10.1038/nature16961

[14] S. Ha and J. Teich, *Handbook of Hardware-Software Codesign*, 1st ed. Springer Publishing Company, Incorporated, 2017. ISBN 9401772681

[15] D. Liu, *Embedded DSP Processor Design: Application Specific Instruction Set Processors*. Morgan Kaufmann, 2008.

[16] G. Box, "Science and Statistics," *Journal of the American Statistical Association*, vol. 71, no. 356, pp. 791–799, 1976. doi: 10.1080/01621459.1976.10480949

[17] R. L.Haupt, *Wireless Communications Systems*. John Wiley and Sons, Inc, 2020.

[18] C. Cox, *An Introduction to 5G*. John Wiley and Sons Ltd, 2021.

[19] T. L. Marzetta, "Noncooperative Cellular Wireless with Unlimited Numbers of Base Station Antennas," *IEEE Transactions on Wireless Communications*, vol. 9, no. 11, pp. 3590–3600, 2010. doi: 10.1109/TWC.2010.092810.091092

[20] E. G. Larsson, O. Edfors, F. Tufvesson, and T. L. Marzetta, "Massive MIMO for next generation wireless systems," *IEEE Communications Magazine*, vol. 52, no. 2, pp. 186–195, 2014. doi: 10.1109/MCOM.2014.6736761

[21] F. Rusek, D. Persson, B. K. Lau, E. G. Larsson, T. L. Marzetta, O. Edfors, and F. Tufvesson, "Scaling Up MIMO: Opportunities and Challenges with Very Large Arrays," *IEEE Signal Processing Magazine*, vol. 30, no. 1, pp. 40–60, 2013. doi: 10.1109/MSP.2011.2178495

[22] E. Björnson, E. G. Larsson, and T. L. Marzetta, "Massive MIMO: ten myths and one critical question," *IEEE Communications Magazine*, vol. 54, no. 2, pp. 114–123, 2016. doi: 10.1109/MCOM.2016.7402270

[23] H. Yang and T. L. Marzetta, "Energy Efficient Design of Massive MIMO: How Many Antennas?" in *2015 IEEE 81st Vehicular Technology Conference (VTC Spring)*, 2015. doi: 10.1109/VTCSpring.2015.7145809 pp. 1–5.

[24] S. Hu, F. Rusek, and O. Edfors, "Beyond Massive MIMO: The Potential of Data Transmission With Large Intelligent Surfaces," *IEEE Transactions on Signal Processing*, vol. 66, no. 10, pp. 2746–2758, 2018. doi: 10.1109/TSP.2018.2816577

[25] R. W. Chang, "Synthesis of band-limited orthogonal signals for multichannel data transmission," *The Bell System Technical Journal*, vol. 45, no. 10, pp. 1775–1796, 1966. doi: 10.1002/j.1538-7305.1966.tb02435.x

[26] S. Weinstein and P. Ebert, "Data Transmission by Frequency-Division Multiplexing Using the Discrete Fourier Transform," *IEEE Transactions on Communication Technology*, vol. 19, no. 5, pp. 628–634, 1971. doi: 10.1109/TCOM.1971.1090705

[27] REINDEER. (2021) REINDEER deliverable D1.1: Use case-driven specifications and technical re-quirements and initial channel model. [Online]. Available: https://doi.org/10.5281/zenodo.5561844

[28] S. Research. 6G The Next Hyper Connected Experience for All. [Online]. Available: https://cdn.codeground.org/nsr/downloads/researchareas/20201201_6G_Vision_web.pdf

[29] H. Viswanathan and P. E. Mogensen, "Communications in the 6G Era," *IEEE Access*, vol. 8, pp. 57 063–57 074, 2020. doi: 10.1109/ACCESS.2020.2981745

[30] E. W. Dijkstra. (1982) Why numbering should start at zero. [Online]. Available: https://www.cs.utexas.edu/~EWD/transcriptions/EWD08xx/EWD831.html

[31] W. Gander, *Algorithms for the QR decomposition*, ser. Seminar für Angewandte Mathematik: Research report, 1980. [Online]. Available: https://books.google.se/books?id=G_UecgAACAAJ

[32] D. S. Watkins, *Fundamentals of matrix computations*. USA: John Wiley and Sons, Inc., 1991. ISBN 0471614149

[33] L. N. Trefethen and D. Bau, *Numerical Linear Algebra*. SIAM, 1997. ISBN 0898713617

[34] M. Attari, L. Ferreira, L. Liu, and S. Malkowsky, "An Application Specific Vector Processor for Efficient Massive MIMO Processing," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 69, no. 9, pp. 3804–3815, 2022. doi: 10.1109/TCSI.2022.3182483

[35] S. Malkowsky, "Massive mimo: Prototyping, proof-of-concept and implementation," Doctoral Thesis (monograph). ISBN 978-91-7895-115-4 Apr. 2019.

[36] D. Wubben, R. Bohnke, V. Kuhn, and K.-D. Kammeyer, "MMSE extension of V-BLAST based on sorted QR decomposition," in *2003 IEEE 58th Vehicular Technology Conference. VTC 2003-Fall (IEEE Cat. No.03CH37484)*, vol. 1, 2003. doi: 10.1109/VETECF.2003.1285069 pp. 508–512 Vol.1.

[37] J. Rodríguez Sánchez, F. Rusek, O. Edfors, and L. Liu, "Distributed and Scalable Uplink Processing for LIS: Algorithm, Architecture, and Design Trade-Offs," *IEEE Transactions on Signal Processing*, vol. 70, pp. 2639–2653, 2022. doi: 10.1109/TSP.2022.3171094

[38] J. Rodríguez Sánchez, F. Rusek, O. Edfors, M. Sarajlić, and L. Liu, "Decentralized Massive MIMO Processing Exploring Daisy-Chain Architecture and Recursive Algorithms," *IEEE Transactions on Signal Processing*, vol. 68, pp. 687–700, 2020. doi: 10.1109/TSP.2020.2964496

[39] J. Rodríguez Sánchez, "Systems with Massive Number of Antennas: Distributed Approaches," Doctoral Thesis (compilation). ISBN 978-91-8039-228-0 2022.

[40] S. L. Brunton and J. N. Kutz, *Data-Driven Science and Engineering: Machine Learning, Dynamical Systems, and Control*. Cambridge University Press, 2019, ch. 1.

[41] J. Chen and V. K. N. Lau, "Multi-stream iterative SVD for massive MIMO communication systems under time varying channels," in *2014 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2014. doi: 10.1109/ICASSP.2014.6854181 pp. 3152–3156.

[42] M. Attari, J. R. Sánchez, and L. Liu, "A Floating-Point 16 $\times$ 16 SVD Accelerator for Beyond-5G Large Intelligent Surfaces," in *IEEE International Midwest Symposium on Circuits and Systems*, 2023. doi: 10.1109/MWSCAS57524.2023.10406077 pp. 967–971.

[43] M. Narwaria and W. Lin, "SVD-Based Quality Metric for Image and Video Using Machine Learning," *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, vol. 42, no. 2, pp. 347–364, 2012. doi: 10.1109/TSMCB.2011.2163391

[44] G. E. Forsythe and P. Henrici, "The Cyclic Jacobi Method for Computing the Principal Values of a Complex Matrix," *Transactions of the American Mathematical Society*, vol. 94, no. 1, pp. 1–23, 1960.

[45] G. Golub and W. Kahan, "Calculating the Singular Values and Pseudo-Inverse of a Matrix," *Journal of the Society for Industrial and Applied Mathematics Series B Numerical Analysis*, vol. 2, no. 2, pp. 205–224, 1965. doi: 10.1137/0702016. [Online]. Available: https://doi.org/10.1137/0702016

[46] G. H. Golub and C. F. van Loan, *Matrix Computations*, 4th ed. The Johns Hopkins University Press, 2013, pp. 486–494. ISBN 1421407949 9781421407944

[47] W. Givens, "Computation of Plain Unitary Rotations Transforming a General Matrix to Triangular Form," *Journal of the Society for Industrial and Applied Mathematics*, vol. 6, no. 1, pp. 26–50, 1958. doi: 10.1137/0106004. [Online]. Available: https://doi.org/10.1137/0106004

[48] G. E. Moore, "Cramming more components onto integrated circuits, reprinted from electronics, volume 38, number 8, april 19, 1965, pp.114 ff." *IEEE Solid-State Circuits Society Newsletter*, vol. 11, no. 3, pp. 33–35, 2006. doi: 10.1109/N-SSC.2006.4785860

[49] G. E. Moore, "Progress in digital integrated electronics [Technical literaiture, Copyright 1975 IEEE. Reprinted, with permission. Technical Digest. International Electron Devices Meeting, IEEE, 1975, pp. 11-13.]," *IEEE Solid-State Circuits Society Newsletter*, vol. 11, no. 3, pp. 36–37, 2006. doi: 10.1109/N-SSC.2006.4804410

[50] P. Brans, "Nvidia's Michael Kagan: Building on AI's "iPhone Moment" to Architect Data Processing's Future," *EE Times Europe*, pp. 11–12, November 2023. [Online]. Available: https://www.eetimes.eu/ee-times-europe-magazine-november-2023/

[51] N. P. Jouppi, C. Young, N. Patil, D. Patterson, G. Agrawal, R. Bajwa, S. Bates, S. Bhatia, N. Boden, A. Borchers *et al.*, "In-datacenter performance analysis of a tensor processing unit," in *2017 ACM/IEEE 44th Annual International Symposium on Computer Architecture (ISCA)*, 2017. doi: 10.1145/3079856.3080246 pp. 1–12.

[52] N. P. Jouppi, C. Young, N. Patil, and D. Patterson, "A domain-specific architecture for deep neural networks," *Commun. ACM*, vol. 61, no. 9, pp. 50–59, aug 2018. doi: 10.1145/3154484. [Online]. Available: https://doi.org/10.1145/3154484

[53] N. Jouppi, G. Kurian, S. Li, P. Ma, R. Nagarajan, L. Nai, N. Patil, S. Subramanian, A. Swing, B. Towles *et al.*, "TPU v4: An Optically Reconfigurable Supercomputer for Machine Learning with Hardware Support for Embeddings," in *Proceedings of the 50th Annual International Symposium on Computer Architecture*, ser. ISCA '23. New York, NY, USA: Association for Computing Machinery, 2023. doi: 10.1145/3579371.3589350. ISBN 9798400700958. [Online]. Available: https://doi.org/10.1145/3579371.3589350

[54] N. P. Jouppi, D. Hyun Yoon, M. Ashcraft, M. Gottscho, T. B. Jablin, G. Kurian, J. Laudon, S. Li, P. Ma, X. Ma *et al.*, "Ten Lessons From Three Generations Shaped Google's TPUv4i : Industrial Product," in *2021 ACM/IEEE 48th Annual International Symposium on Computer Architecture (ISCA)*, 2021. doi: 10.1109/ISCA52012.2021.00010 pp. 1–14.

[55] M. Mahdavi, O. Edfors, V. Öwall, and L. Liu, "A Low Latency FFT/IFFT Architecture for Massive MIMO Systems Utilizing OFDM Guard Bands," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 66, no. 7, pp. 2763–2774, 2019. doi: 10.1109/TCSI.2019.2896042

[56] A. Burg, M. Borgmann, M. Wenk, M. Zellweger, W. Fichtner, and H. Bolcskei, "VLSI implementation of MIMO detection using the sphere decoding algorithm," *IEEE Journal of Solid-State Circuits*, vol. 40, no. 7, pp. 1566–1577, 2005. doi: 10.1109/JSSC.2005.847505

[57] M. Kuhn, S. Moser, O. Isler, F. Gurkaynak, A. Burg, N. Felber, H. Kaeslin, and W. Fichtner, "Efficient ASIC implementation of a real-time depth mapping stereo vision system," in *2003 46th Midwest Symposium on Circuits and Systems*, vol. 3, 2003. doi: 10.1109/MWSCAS.2003.1562575 pp. 1478–1481 Vol. 3.

[58] L. Liu, J. Zhu, Z. Li, Y. Lu, Y. Deng, J. Han, S. Yin, and S. Wei, "A Survey of Coarse-Grained Reconfigurable Architecture and Design: Taxonomy, Challenges, and Applications," *ACM Comput. Surv.*, vol. 52, no. 6, oct 2019. doi: 10.1145/3357375. [Online]. Available: https://doi.org/10.1145/3357375

[59] H. Goldstine and A. Goldstine, "The Electronic Numerical Integrator and Computer (ENIAC)," *IEEE Annals of the History of Computing*, vol. 18, no. 1, pp. 10–16, 1996. doi: 10.1109/85.476557

[60] S. M. Trimberger, "Three Ages of FPGAs: A Retrospective on the First Thirty Years of FPGA Technology," *Proceedings of the IEEE*, vol. 103, no. 3, pp. 318–331, 2015. doi: 10.1109/JPROC.2015.2392104

[61] W. S. Carter, I. Duong, R. R. Freman, H. Hsieh, J. Y. Ja, J. E. Mahoney, N. T. Ngo, and S. L. Sac, "A user programmable reconfigurable logic array," 1986. [Online]. Available: https://api.semanticscholar.org/CorpusID:60420567

[62] Mirsky and DeHon, "MATRIX: a reconfigurable computing architecture with configurable instruction distribution and deployable resources," in *1996 Proceedings IEEE Symposium on FPGAs for Custom Computing Machines*, 1996. doi: 10.1109/FPGA.1996.564808 pp. 157–166.

[63] R. Prabhakar, Y. Zhang, D. Koeplinger, M. Feldman, T. Zhao, S. Hadjis, A. Pedram, C. Kozyrakis, and K. Olukotun, "Plasticine: A reconfigurable architecture for parallel patterns," in *2017 ACM/IEEE 44th Annual International Symposium on Computer Architecture (ISCA)*, 2017. doi: 10.1145/3079856.3080256 pp. 389–402.

[64] Y. Zhang, N. Zhang, T. Zhao, M. Vilim, M. Shahbaz, and K. Olukotun, "SARA: Scaling a Reconfigurable Dataflow Accelerator," in *2021 ACM/IEEE 48th Annual International Symposium on Computer Architecture (ISCA)*, 2021. doi: 10.1109/ISCA52012.2021.00085 pp. 1041–1054.

[65] J. Weng, S. Liu, V. Dadu, Z. Wang, P. Shah, and T. Nowatzki, "DSAGEN: Synthesizing Programmable Spatial Accelerators," in *2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA)*, 2020. doi: 10.1109/ISCA45697.2020.00032 pp. 268–281.

[66] B. Bougard, B. De Sutter, S. Rabou, D. Novo, O. Allam, S. Dupont, and L. Van der Perre, "A Coarse-Grained Array based Baseband Processor for 100Mbps+ Software Defined Radio," in *2008 Design, Automation and Test in Europe*, 2008. doi: 10.1109/DATE.2008.4484763 pp. 716–721.

[67] J. Fowers, K. Ovtcharov, M. Papamichael, T. Massengill, M. Liu, D. Lo, S. Alkalay, M. Haselman, L. Adams, M. Ghandi *et al.*, "A configurable cloud-scale DNN processor for real-time AI," in *Proceedings of the 45th Annual International Symposium on Computer Architecture*, ser. ISCA '18. IEEE Press, 2018. doi: 10.1109/ISCA.2018.00012. ISBN 9781538659847 pp. 1–14. [Online]. Available: https://doi.org/10.1109/ISCA.2018.00012

[68] C. Kim, M. Chung, Y. Cho, M. Konijnenburg, S. Ryu, and J. Kim, "ULP-SRP: Ultra low power Samsung Reconfigurable Processor for biomedical applications," in *2012 International Conference on Field-Programmable Technology*, 2012. doi: 10.1109/FPT.2012.6412157 pp. 329–334.

[69] S. Ciricescu, R. Essick, B. Lucas, P. May, K. Moat, J. Norris, M. Schuette, and A. Saidi, "The reconfigurable streaming vector processor (RSVP/spl trade/)," in *Proceedings. 36th Annual IEEE/ACM International Symposium on Microarchitecture, 2003. MICRO-36.*, 2003. doi: 10.1109/MICRO.2003.1253190 pp. 141–150.

[70] J. E. Thornton, "Parallel operation in the control data 6600," in *Proceedings of the October 27-29, 1964, Fall Joint Computer Conference, Part II: Very High Speed Computer Systems*, ser. AFIPS '64 (Fall, part II). New York, NY, USA: Association for Computing Machinery, 1964. doi: 10.1145/1464039.1464045. ISBN 9781450378888 pp. 33–40. [Online]. Available: https://doi.org/10.1145/1464039.1464045

[71] G. Radin, "The 801 minicomputer," in *Proceedings of the First International Symposium on Architectural Support for Programming Languages and Operating Systems*, ser. ASPLOS I. New York, NY, USA: Association for Computing Machinery, 1982. doi: 10.1145/800050.801824. ISBN 0897910664 pp. 39–47. [Online]. Available: https://doi.org/10.1145/800050.801824

[72] A. Waterman, Y. Lee, D. A. Patterson, and K. Asanović, "The RISC-V Instruction Set Manual, Volume I: User-Level ISA, Version 2.0," Tech. Rep. UCB/EECS-2014-54, May 2014. [Online]. Available: http://www2.eecs.berkeley.edu/Pubs/TechRpts/2014/EECS-2014-54.html

[73] A. Waterman, "Design of the RISC-V Instruction Set Architecture," Ph.D. dissertation, EECS Department, University of California, Berkeley, Jan 2016. [Online]. Available: http://www2.eecs.berkeley.edu/Pubs/TechRpts/2016/EECS-2016-1.html

[74] D. A. Patterson and D. R. Ditzel, "The case for the reduced instruction set computer," *SIGARCH Comput. Archit. News*, vol. 8, no. 6, pp. 25–33, oct 1980. doi: 10.1145/641914.641917. [Online]. Available: https://doi.org/10.1145/641914.641917

[75] Intel. (2024) Intel 64 and IA-32 Architectures Software Developer Manuals. [Online]. Available: https://www.intel.com/content/www/us/en/developer/articles/technical/intel-sdm.html

[76] Intel. (2024) Intel 64 and IA-32 Architectures Software Developer's Manual Combined Volumes: 1, 2A, 2B, 2C, 2D, 3A, 3B, 3C, 3D, and 4. [Online]. Available: https://cdrdv2.intel.com/v1/dl/getContent/671200

[77] NVIDIA. (2023) NVIDIA ADA GPU ARCHITECTURE. [Online]. Available: https://images.nvidia.com/aem-dam/Solutions/Data-Center/l4/nvidia-ada-gpu-architecture-whitepaper-V2.02.pdf

[78] E. Strohmaier, J. Dongarra, H. Simon, M. Meuer, and H. Meuer. (2024) Top 500 - The List. [Online]. Available: https://top500.org/

[79] A. Fauth, J. Van Praet, and M. Freericks, "Describing instruction set processors using nML," in *Proceedings the European Design and Test Conference. ED and TC 1995*, 1995. doi: 10.1109/EDTC.1995.470354 pp. 503–507.

[80] A. Wolfe, M. Breternitz, C. Stephens, A. Ting, D. Kirk, R. Bianchini, and J. Shen, "The White Dwarf: a high-performance application-specific processor," in *[1988] The 15th Annual International Symposium on Computer Architecture. Conference Proceedings*, 1988. doi: 10.1109/ISCA.1988.5231 pp. 212–222.

[81] M. Gschwind, "Instruction set selection for ASIP design," in *Proceedings of the Seventh International Workshop on Hardware/Software Codesign (CODES'99) (IEEE Cat. No.99TH8450)*, 1999. doi: 10.1145/301177.301187 pp. 7–11.

[82] Q. Dinh, D. Chen, and M. D. F. Wong, "Efficient ASIP design for configurable processors with fine-grained resource sharing," in *Proceedings of the 16th International ACM/SIGDA Symposium on Field Programmable Gate Arrays*, ser. FPGA '08. New York, NY, USA: Association for Computing Machinery, 2008. doi: 10.1145/1344671.1344687. ISBN 9781595939340 pp. 99–106. [Online]. Available: https://doi.org/10.1145/1344671.1344687

[83] Shahabuddin, Shahriar and Mämmelä, Aarne and Juntti, Markku and Silvén, Olli, "ASIP for 5G and Beyond: Opportunities and Vision," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 68, no. 3, pp. 851–857, 2021. doi: 10.1109/TCSII.2021.3050785

[84] Synopsys. ASIP Designer. [Online]. Available: https://www.synopsys.com/dw/ipdir.php?ds=asip-designer

[85] Cadence. Tensilica. [Online]. Available: https://www.cadence.com/en_US/home/tools/silicon-solutions/compute-ip/tensilica-xtensa-controllers-and-extensible-processors.html

[86] Codasip. Codasip. [Online]. Available: https://codasip.com/

[87] K. Hepola, J. Multanen, and P. Jääskeläinen, "OpenASIP 2.0: Co-Design Toolset for RISC-V Application-Specific Instruction-Set Processors," in *2022 IEEE 33rd International Conference on Application-specific Systems, Architectures and Processors (ASAP)*, 2022. doi: 10.1109/ASAP54787.2022.00034 pp. 161–165.

[88] (2024) OpenASIP - Open Application-Specific Instruction-set Processor toolset. [Online]. Available: https://github.com/cpc/openasip

[89] L. Van der Perre, L. Liu, and E. G. Larsson, "Efficient DSP and Circuit Architectures for Massive MIMO: State of the Art and Future Directions," *IEEE Transactions on Signal Processing*, vol. 66, no. 18, pp. 4717–4736, 2018. doi: 10.1109/TSP.2018.2858190

[90] T. Nowatzki, V. Gangadhan, K. Sankaralingam, and G. Wright, "Pushing the limits of accelerator efficiency while retaining programmability," in *2016 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, 2016. doi: 10.1109/HPCA.2016.7446051 pp. 27–39.

[91] Y. Liu, L. Liu, and V. Öwall, "Architecture design of a memory subsystem for massive MIMO baseband processing," *IEEE Trans. Very Large Scale Integration Systems*, vol. 25, no. 10, pp. 2976–2980, 2017. doi: 10.1109/TVLSI.2017.2732062

[92] A. Doyle, *The Five Orange Pips and Other Cases*, ser. The Penguin English Library. Penguin Books Limited, 2012. ISBN 9780141974668. [Online]. Available: https://books.google.se/books?id=UJLeA50QxPEC

[93] J. Vieira, E. Leitinger, M. Sarajlic, X. Li, and F. Tufvesson, "Deep convolutional neural networks for massive MIMO fingerprint-based positioning," in *2017 IEEE 28th Annual International Symposium on Personal, Indoor, and Mobile Radio Communications (PIMRC)*. IEEE, 2017.

[94] S. De Bast and S. Pollin, "MaMIMO CSI-Based Positioning using CNNs: Peeking inside the Black Box," in *2020 IEEE International Conference on Communications Workshops (ICC Workshops)*, 2020. doi: 10.1109/ICCWorkshops49005.2020.9145412 pp. 1–6.

[95] M. Attari, J. Rodríguez Sánchez, L. Liu, and S. Malkowsky, "An Application Specific Vector Processor for CNN-Based Massive MIMO Positioning," in *2021 IEEE International Symposium on Circuits and Systems (ISCAS)*, 2021. doi: 10.1109/ISCAS51556.2021.9401528 pp. 1–5.

APPENDICES

A

Code Examples

In this appendix, C code examples are given for selected algorithms. The intrinsic functions, which guide the compiler to map the function name to appropriate hardware datapaths, are distinguished in code by appending `_c` at the end.

```
void cholesky(vcbfloat16* in, vcbfloat16* out)
{
    for(int j = 0; j < 16; j++)
    {
        vcbfloat16 s = read_col_shuffled_c(&in[j]);

        for (int k = 0; k <= j-1; k++)
        {
            vcbfloat16 tmp =
                read_col_shuffled_c(&out[k]) *
                read_col_shuffled_c(&out[k])[j];
            s = s - tmp;
        }

        cbfloat16 one_by_sqrt = inverse_sqrt_c(s[j]);
        store_col_shuffled_c(&out[j], s * one_by_sqrt, 15, j);
    }
}
```

Figure A.1: C code for Cholesky.

```
void qrd(
    vcbfloat16* gramian_mat, vcbfloat16* r_mat, vcbfloat16* q_mat
)
{
    for (int i = 0; i < 16; i++)
    {
        vcbfloat16 vi = read_col_shuffled_c(&gramian_mat[i]);
        cbfloat16 norm = vec_norm_c(vi, vi);
        vcbfloat16 r = read_row_shuffled_c(&r_mat[i]);
        r = vec_index_modify_c(r, i, norm);
        cbfloat16 inorm = inverse_c(norm);
        vcbfloat16 qi = vec_mul_c(vi, inorm);
        store_col_shuffled_c(&q_mat[i], qi);
        for (int j = i+1; j < 16; j++)
        {
            vcbfloat16 vj = read_col_shuffled_c(&gramian_mat[j]);
            cbfloat16 dot = vec_dot_c(qi, vj);
            r = vec_index_modify_c(r, j, dot);

            vj = vj - (qi * r[j]);
            store_col_shuffled_c(&gramian_mat[j], vj);
        }
        store_row_shuffled_c(&r_mat[i], r);
    }
}
```

Figure A.2: C code for QRD.

```
void ext_qrd(vcbfloat16* q_mat, vcbfloat16* rinv_mat)
{
    for (int i = 0; i < 16; i++)
    {
        vcbfloat16 qi = read_col_shuffled_c(&q_mat[i]);
        vcbfloat16 ii = read_col_shuffled_c(&rinv_mat[i]);
        vcbfloat16 tmp = qi + ii;
        cbfloat16 dot = vec_dot_c(tmp, tmp);
        cbfloat16 inorm = inverse_sqrt_c(dot);
        qi = vec_mul_c(qi, inorm);
        ii = vec_mul_c(ii, inorm);
        store_col_shuffled_c(&q_mat[i], qi);
        store_col_shuffled_c(&rinv_mat[i], ii);

        for (int j = i+1; j < 16; j++)
        {
            vcbfloat16 qj = read_col_shuffled_c(&q_mat[j]);
            vcbfloat16 ij = read_col_shuffled_c(&rinv_mat[j]);
            cbfloat16 r_ij =
                scalar_add_c(vec_dot_c(qj, qi), vec_dot_c(ij, ii));
            qj = qj - qi * r_ij;
            ij = ij - ii * r_ij;
            store_col_shuffled_c(&q_mat[j], qj);
            store_col_shuffled_c(&rinv_mat[j], ij);
        }
    }
}
```

Figure A.3: C code for extended QRD.

```
void matrix_mul_16x16_by_16x16(
    vcbfloat16* a_in, vcbfloat16* b_in, vcbfloat16* mul_out
)
{
    for (int ra = 0; ra < 16; ra++)
    {
        vcbfloat16 sum = 0;
        vcbfloat16 a = read_row_shuddled_c(&a_in[ra]);
        for (int rb = 0; rb < 16; rb++)
        {
            vcbfloat16 b = read_row_shuddled_c(&b_in[rb]);
            sum = vec_mac_c(sum, b, index_2nd_c(a, rb));
        }
        store_row_shuddled_c(&mul_out[ra], sum);
    }
}
```

Figure A.4: C code for 16×16 by 16×16 matrix multiplication.

```
void gramian(vcbfloat16* h_in, vcbfloat16* gr_out)
{
    for (int col = 0; col < 16; col++)
    {
        vcbfloat16 sum = 0;
        for (int row = 0; row < 128; row++)
        {
            vcbfloat16 a = read_row_shuufled_c(&h[row]);
            sum = vec_mac_c(sum, conj_index_2nd_c(a, col));
        }
        store_row_shuufled_c(&gr_out[col], (vcbfloat16)sum);
}
```

Figure A.5: C code for Gramian calculation.

```
void matrix_mul_gnereal(
    vcbfloat16* a_in, vcbfloat16* b_in, vcbfloat16* mul_out,
    int num_rows_a, int num_cols_a,
    int num_rows_b, int num_cols_b
)
{
    int num_a_row_blocks = num_rows_a >> 4;
    int num_a_col_blocks = num_cols_a >> 4;
    int num_b_col_blocks = num_cols_b >> 4;

    const int N = NUM_VEC_LANES;

    for (
        int b_col_block_index = 0;
        b_col_block_index < num_b_col_blocks; b_col_block_index++
    )
    {
        for (
            int a_row_index = 0;
            a_row_index < num_rows_a; a_row_index++
        )
        {
            vcbfloat16 sum = 0;

            for (
                int a_col_block_index = 0;
                a_col_block_index < num_a_col_blocks;
                a_col_block_index++
            )
            {
                int a_index =
                    a_row_index +
                    (a_col_block_index * N * num_a_row_blocks);
                vcbfloat16 a = read_row_shuffled_c(&a_in[a_index]);

                for (int rb = 0; rb < N; rb++)
                {
                    int b_index =
                        rb + (a_col_block_index * N) +
                        (b_col_block_index * num_a_col_blocks * N);
                    vcbfloat16 b = read_row_shuffled_c(&b_in[b_index]);
                    sum = vec_mac_c(sum, b, index_2nd_c(a, rb));
                }
            }

            int mul_index = a_row_index +
                (b_col_block_index * num_a_col_blocks * N);
            store_row_shuffled_c(&mul_out[mul_index], sum);
        }
    }
}
```

Figure A.6: C code for general matrix multiplication.