Characterization and Acceleration of High Performance Compute Workloads

Citation for published version (APA):
Corda, S. (2022). *Characterization and Acceleration of High Performance Compute Workloads*. [Phd Thesis 1 (Research TU/e / Graduation TU/e), Electrical Engineering]. Eindhoven University of Technology.

Document status and date:
Published: 19/12/2022

Document Version:
Publisher’s PDF, also known as Version of Record (includes final page, issue and volume numbers)

Please check the document version of this publication:

• A submitted manuscript is the version of the article upon submission and before peer-review. There can be important differences between the submitted version and the official published version of record. People interested in the research are advised to contact the author for the final version of the publication, or visit the DOI to the publisher's website.
• The final author version and the galley proof are versions of the publication after peer review.
• The final published version features the final layout of the paper including the volume, issue and page numbers.

Link to publication

General rights
Copyright and moral rights for the publications made accessible in the public portal are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.

• Users may download and print one copy of any publication from the public portal for the purpose of private study or research.
• You may not further distribute the material or use it for any profit-making activity or commercial gain
• You may freely distribute the URL identifying the publication in the public portal.

If the publication is distributed under the terms of Article 25fa of the Dutch Copyright Act, indicated by the “Taverne” license above, please follow below link for the End User Agreement:

www.tue.nl/taverne

Take down policy
If you believe that this document breaches copyright please contact us at:

openaccess@tue.nl

providing details and we will investigate your claim.

Download date: 04. Oct. 2023

Characterization and Acceleration of High Performance Compute Workloads

THESIS

ter verkrijging van de graad van doctor aan de Technische Universiteit Eindhoven, op gezag van de rector magnificus prof.dr.ir. F.P.T. Baaijens, voor een commissie aangewezen door het College voor Promoties, in het openbaar te verdedigen op maandag 19 december 2022 om 11:00 uur

door

Stefano Corda

geboren te Moncalieri, Italië

Dit proefschrift is goedgekeurd door de promotor en de samenstelling van de commissie is als volgt:

Voorzitter:	prof.dr.ir. A. M. J. Koonen
Promotoren:	prof.dr. H. Corporaal
	prof.dr.ir. A. Kumar (Technische Universität Dresden)
Copromotor:	dr.ir. R. Jordans
Promotiecommissieleden:	dr.ir. A.B.J. Kokkeler (University of Twente)
	prof.dr.ing. D. Goehringer (Technische Universität Dresden)
	prof.dr.ir. P.H.N. de With
Adviseur:	dr. J. Romein (ASTRON)
	dr. A.J. Awan (Ericsson)

Het onderzoek dat in dit proefschrift wordt beschreven is uitgevoerd in overeenstemming met de TU/e Gedragscode Wetenschapsbeoefening.

Characterization and Acceleration of High Performance Compute Workloads

Stefano Corda

Doctorate committee:

prof.dr. H. Corporaal	Eindhoven University of Technology, 1st promotor
prof.dr.ir. A. Kumar	Dresden Technical University, 2nd promotor
dr.ir. R. Jordans	Eindhoven University of Technology copromotor
prof.dr.ir. A. M. J. Koonen	Eindhoven University of Technology, chairman
prof.dr.ir. D. Göhringer	Dresden Technical University
prof.dr.ir. A. B. J. Kokkeler	University of Twente
prof.dr.ir. P.H.N. de With	Eindhoven University of Technology
dr. J. R. Romein	ASTRON
dr. A. J. Awan	Ericsson Sweden

This work was supported in part by the European Commission by the Horizon 2020 Program for the Project “Near-Memory Computing (NeMeCo)” through the Marie Sklodowska-Curie Innovative Training Networks European Industrial Doctorate under Project 676240.

© Copyright 2022, Stefano Corda
All rights reserved. Reproduction in whole or in part is prohibited without the written consent of the copyright owner.

ISBN: 978-90-386-5633-5

A catalogue record is available from the Eindhoven University of Technology Library.

Acknowledgements

This doctoral thesis finally concludes a long period of studies and research. The Ph.D. journey improved me both professionally and personally. This section acknowledges those who supported and helped me directly and indirectly during my Ph.D.

I am deeply grateful to my supervisors, Henk, Akash, and Roel, for their invaluable advice, continuous support, and patience during my Ph.D. study. Thanks Henk, for the rigorous feedback on papers and for teaching me how to conduct impactful research. I am grateful to Akash for allowing me to join his research group and supporting me in writing funding proposals. Thanks Roel, for your precious guidance and assistance in supervising Master' students, which I would also like to thank.
I would like to thank the Committee for reading the thesis and providing valuable feedback to improve the quality of the manuscript.
This work was only possible with inspiring and brilliant collaborators. I am grateful to Ahsan for providing his immense knowledge of application profiling and hardware. I had the pleasure of collaborating with researchers at ASTRON over these years: Albert-Jan, John, and Bram. Thanks you all for your support and, in particular, Bram for the innumerable long discussions on IDG and HPC. Special thanks to my colleagues at the Electronics System Group, the Parse, and the Nemeco team at TU/e and the Processor Design Chair at Dresden Technical University for providing an enjoyable working environment.

A big thanks to my family, in-law family, and friends for the support and blessing. I would like to express my deepest gratitude to my parents, Pia and Antonello. Thanks for growing me with the desire to learn and improve myself. Thanks also for your love and support during all my life steps and studies.
Above all, I would like to thank my loving wife, Ilenia, for always supporting me throughout all moments of my Ph.D and encouraging me to do my best on every occasion. Knowing that you were at my side made everything easier. Thanks for your patience when I was stressed, for the free time spent on this thesis, and for making sacrifices while living in different countries. I am deeply grateful for your being my partner for 10 years and for many more yet to come.

Abstract

Modern big-data workloads have demanding performance requirements. This leads to compute and memory bottlenecks. These applications comprise, among others, radio-astronomical imaging, machine learning and bioinformatics algorithms. For instance, the future generation of radio-telescopes, such as the Square Kilometre Array (SKA), will have to process a massive quantity of data (in the order of Terabytes per second per antenna) using high-performance computing systems (in the order of Exaflop per second) with high energy efficiency.
With the demise of Dennard scaling and slowing down of Moore's law, computing performance is hitting a plateau. Furthermore, the improvements in memory and processor technology have grown at different speeds, infamously termed the memory wall. These challenges make it difficult to meet the requirements of such demanding applications.
A promising solution to bridge this gap is represented by High-Performance Computing (HPC), which uses modern architectures such as multi-core CPUs, GPUs, and FPGAs to accelerate workloads by optimizing the code to exploit performance close to their limits. Furthermore, among today's emerging computing paradigms, Near-Memory Computing (NMC) rises. NMC is a data-centric computation approach that performs the computation near the memory, avoids data movements that characterize classical compute-centric systems, and potentially is a candidate for high-performance computing.
With the advent of numerous emerging computing systems, it has become crucial to characterize applications for highlighting performance bottlenecks and optimization opportunities. Moreover, algorithm optimization and acceleration are key factors for providing high performance on modern computing systems. However, contemporary workloads do not perform equally on different systems, e.g., GPUs and FPGAs. This leads to a careful selection of application-domain architectures and optimizations. To overcome the abovementioned issue, we need to investigate application characterization techniques aided with machine learning for efficient offloading decisions and optimize the performance bottlenecks in radio-astronomical imaging applications on heterogeneous architectures.
The thesis focuses on providing key contributions toward application profiling and optimization for high-performance computing systems. It extends the state-of-the-art Platform-Independent Software Analysis (PISA) with metrics concerning memory and parallelism relevant to NMC. The metrics include memory entropy, spatial locality, data-level, and basic-block-level parallelism. By profiling a set

of representative applications and correlating the metrics with the application's performance on a simulated NMC system, it demonstrates these additional metrics improve state-of-the-art tools in identifying applications suitable for NMC architectures.

Since hardware-independent analysis is expensive in terms of computation time and resources, the thesis suggests employing an ensemble machine learning model together with hardware-dependent application analysis, which reduces the prediction time up to 3 orders of magnitude compared with the state-of-the-art.

While the previous contributions employ the benchmark methodology, the thesis also focuses on the real-world use case of radio-astronomical imaging, where CPI (Clock Per Instruction) breakdown analysis on modern CPUs identifies large 2D FFTs and Gridder to be a performance bottleneck. It presents an NMC accelerator for 2D FFT computation and shows its implementation on FPGA outperforms the CPU counterpart and performs comparably to a high-end GPU.

To improve the performance of Gridder, it exploits reduced precision acceleration contrary to the usual practice of employing high-precision computations in radio astronomy imaging. Reduced-precision analysis shows that precision must be selected carefully. It presents the first reduced precision accelerator for Gridder, employing custom floating-point data types on FPGA. The prototype outperforms a CPU and keeps up with a GPU with similar peak performance and lithography technology.

To summarize, the thesis's contributions are: 1) PISA-NMC, a hardware-agnostic tool to characterize applications with metrics directed towards near-memory computing; 2) NMPO, an ensemble machine learning framework for offloading prediction on NMC systems; 3) an evaluation of modern HPC architectures, including NMC, for accelerating radio-astronomical large 2D FFTs; and 4) an in-depth reduced precision analysis and the first custom floating-point accelerator for radio-astronomical imaging. Together, these contributions push the state-of-the-art of application characterization and architectural optimization forwards, focusing on real-world applications such as radio-astronomical imaging.

Contents

1 Introduction

Modern scientific applications have high-performance requirements. Workloads are from disparate field such as machine learning [1], genomics [2], hydrodynamics [3], radio-astronomy [4], etc. Radio-astronomy algorithms represent a noteworthy example. Indeed, the Square Kilometre Array (SKA) [5], which is in the first construction phase during this decade, will be the biggest radio-telescope in the world, with thousands of antennas and hosted in South Africa and Australia. A subset of tasks of the SKA is focused on extracting real-time information of transients, in other words detecting interesting objects in the sky like pulsar, and imaging the digitized signal of the sky in images. A radio-telescope of such dimension will produce a large amount of data traffic in the order of TeraBytes/s, and each computing facility will have to provide computing capabilities in the order of ExaFLOP/s [6].
To accommodate such high-performance requirements, innovative approaches need to be researched. Moreover, hardware limitations raise in the past years. In particular, memory technology could not keep up with processing elements technology in terms of speed and energy consumption; this problem is typically called the `Memory Wall` [7]. In addition, chip manufacturers have difficulties in following Moore's [8] and Dennard's [9] Laws. Indeed, it is not anymore possible to shrink the size of transistors like in the past. To overcome these issues, new technologies and computational paradigms have been proposed. For instance, new 3D stacked memories such as High Bandwidth Memory (HBM) [10] and Hybrid Memory Cube (HMC) [11] are employed in current GPUs, and FPGAs generation [12,13]. These innovations are slowly affecting also the CPU architectures; for example the new AMD's 3D-V cache (see *Figure 1.1*). Indeed, 3D memory technology

allows to have large caches with faster interconnect to the cores. Thanks to these new technologies, computational paradigms such as Near-Memory Computing (NMC) [14] have re-gained interest.

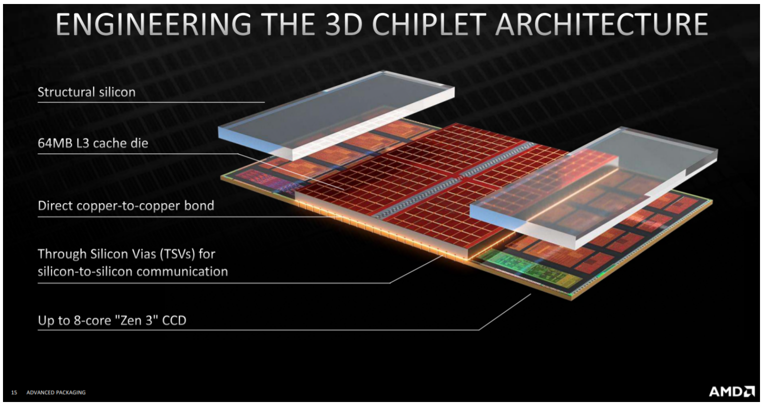

ENGINEERING THE 3D CHIPLET ARCHITECTURE
Structural silicon
64MB L3 cache die
Direct copper-to-copper bond
Through Silicon Vias (TSVs) for silicon-to-silicon communication
Up to 8-core "Zen 3" CCD
15 ADVANCED PACKAGING
AMD

Figure 1.1: *AMD 3D-V Cache technology [15, 16]. The CPU shown is equipped with 3D stacked L3 cache providing higher memory density and lower latency compared to traditional 2D-cache-equipped CPUs.*

The trend of modern computing systems is directed towards heterogeneity. This simply means that supercomputers will consist of different nodes with different resource inter- and intra- nodes. Indeed, certain applications are better suited for certain architecture, such as CPUs [17] and GPUs [18]. Most of the future machines will host domain-specific accelerators, which are basically optimized for specific algorithms. The acquisitions of Altera and Xilinx by respectively Intel [19], and AMD [20] make FPGAs break into the HPC world, delivering promising competitive platforms for scientific computation with respect to GPUs. From one side, heterogeneity allows optimal hardware for the considered application. On the other, it complicates and creates challenges in characterizing workloads, optimizing them for the specific architectures, on which architecture they should be offloaded to, etc. In particular, the challenges regarding application characterization, offloading and optimization on the domain-specif accelerator will be treated in *Section 1.1*. Then we present the thesis' problem statement in *Section 1.2*, contributions in *Section 1.3* and the thesis structure in *Section 1.4*.

1.1 Challenges

The complexity of modern algorithms and computing systems will present difficult challenges to researchers and developers. In particular, here we present the challenges related to application characterization and offloading and the optimization on domain-specific accelerators.

Application characterization and offloading

One of the most difficult challenges is characterizing applications and deciding if and where to offload them to maximize performance. Heterogeneity makes this even more complex. Indeed, profiling and characterizing applications is not standardized, and many tools and characterization metrics have been proposed in the past decades. Furthermore, this becomes even more complex if the target is a future architecture, which can only be simulated [21].
A fair amount of research has been spent on abstracting application characterization from hardware details and focusing only on application properties [22, 23]. However, when targeting novel computation paradigms, new metrics needs to be researched to detect application hotspots that can take advantage of new hardware features. Indeed, this is still an open challenge for paradigms such as near-memory computing [14].

Application characterization should provide enough information to select the most profitable hardware for a specific application. While automatic methodologies such as compilers, which determine application hotspot and provide code optimization, have been introduced [24], they effectively work on simple and standard micro-kernels such as linear algebra (BLAS). Nonetheless, modern workloads contain non-standard computations, which are difficult to detect automatically. Therefore, possible solutions to determine offloading decisions are analyzing profiling results to determine the most relevant characterization metrics or machine-learning prediction model to determine offloading choices [21].

Domain-specific optimization

To accomodate the above mentioned high-performance requirements, powerful machines need to be developed and alternative computing approaches must be researched. Undoubtedly, modern supercomputers can achieve ExaFLOP/s performance but consume a large amount of energy.
Table 1.1 shows the first 5 positions of the TOP500 list [25]. The execution of HPCG [26] on the most powerful top500 supercomputers achieve less than 5% of their peak performance. The waste of power consumption of these machines could be used to sustain the energy demand of a small city [27].
With new architectures entering the domain, such as FPGAs, it becomes even more critical to understand how different platforms behave and can be employed to obtain the best performance in terms of time to solution, but more importantly, energy efficiency.
To achieve this goal, a good knowledge of how the computing architectures are designed is needed. Moreover, to achieve even higher performance and lower energy, a trade-off on the result quality may also be taken into account, e.g., by exploiting approximate computing [28]. However, as we will discuss in *Chapter 6* it strictly depends on applications and datasets. Indeed, radio-astronomical

Table 1.1: *Latest (November 2022) results of the 5 faster supercomputers from top500 list [25]. Frontier is the first machine that surpasses the ExaFlop in raw performance. The table also reports the fraction of peak performance achieved by running HPCG [26], a standard benchmark to assess performance on supercomputers: most of the supercomputers are below 5%.*

Rank	**System**	**Rpeak (PFLOP/s)**	**Power (MW)**	**HPCG %**
1	Frontier	1685.65	21.10	1.00
2	Fugaku	537.21	29.90	3.62
3	LUMI	428.70	2.94	1.10
4	Leonardo	255.75	7.44	1.47
5	Summit	200.79	10.10	1.97

imaging applications typically need high precision, which may vary slightly based on the datasets.

1.2 Problem Statement

With the rising of modern real-world applications in various domains such as radio astronomy, genomics, artificial intelligence, and hydrodynamics that have demanding computational and memory requirements, it is crucial to understand which architecture can better support them and how to optimize them for specific hardware.
Indeed, it is challenging to determine which application sections, typically named kernels, should be offloaded to accelerators. This can be done in very different ways, and it is usually challenging for emerging technologies. More precisely, with the advent of new technologies such as High Bandwidth Memory (HBM) and Hybrid Memory Cube (HMC), new architectures and computational paradigms have been proposed. An example is near-memory computing which shifts the computation near the memory, avoiding the movement through the cache hierarchy. Additionally, competing architectures such as GPU and FPGA recently support this particular technology. This makes it difficult to understand when an architecture is beneficial and when to offload particular kernels.
Even though modern compilers can optimize particularly well applications code for multiple platforms, they are often limited to a few well-known kernels such as linear algebra (BLAS) [29]. However, modern applications in radio astronomical imaging and hydrodynamics include linear-algebra kernels and uncommon computations such as sine/cosine and floating-point operations with particular patterns. It is challenging to map and optimize these computational tasks to new accelerator architectures. Indeed, typically for every architecture, it is usually necessary to

employ specific code grammar to obtain high-performance code, e.g., CUDA for NVIDIA GPUs, Xilinx High-Level Synthesis Pragmas for Xilinx FPGAs.

More precisely, we need to understand *which application characterization methodology would provide sufficient information to make offloading decisions on heterogeneous high-performance computing systems.* Furthermore, *can machine learning improve the above-mentioned offloading decisions?* After determining bottlenecks and taking offloading decisions, *which accelerators and optimization can maximize the computing and energy efficiency of real-world applications such as radio-astronomical imaging?*

1.3 Thesis contributions

This thesis provides techniques and demonstrations of application characterization and optimization by targeting both emerging and state-of-the-art high-performance systems and real-world workloads such as radio-astronomical imaging. We show how application profiling techniques can be used efficiently to detect possible application bottlenecks and individuate appealing optimization for improving performance.
This thesis makes the following four contributions:

1. *Platform-Independent Software Analysis for Near-Memory Computing*, is a hardware-independent tool capable of detecting application features related to data and memory parallelism. With this tool, we extend the application characterization possibilities for emerging computational paradigms such as near-memory computing, and we show its relationship with near-memory computing performance (*Chapter 3*) [30, 31].

2. Emerging high-performance computational paradigm typically employs expensive simulation to understand the suitability for offloading. While machine learning can reduce the amount of hardware simulation to the training set, the current state-of-the-art uses costly hardware-independent characterization to predict NMC offloading suitability. The *Near Memory computing Profiling and Offloading* framework demonstrates that the offloading suitability prediction can drastically benefit from using hardware-dependent characterization (*Chapter 4*) [32].

3. Modern radio-telescope will need to deal with large datasets for detailed sky images. This introduces a bottleneck in the radio-astronomical imaging pipeline: the two-dimensional Fast Fourier Transform. We propose a detailed application characterization and evaluation on state-of-the-art systems including near-memory computing (*Chapter 5*) [33].

4. Radio-astronomical imaging requires high-performance and energy-efficient computing. Reduced precision is a technique that employs reduced precision

data types to improve application performance and saving energy. In this context, we propose for the first time an in-depth reduced-precision analysis and custom accelerator for this application domain. We also provide guidelines for optimizing applications on In Xilinx Vitis targeting Xilinx Alveo FPGAs and important lessons learned (*Chapter 6*) [34].

To summarize, this thesis provides mechanisms and methodologies to ease application profiling to detect application hotspots. Moreover, we evaluate different state-of-the-art architectures showing advantages and disadvantages in modern scientific computing, with a particular focus on radio-astronomical imaging workloads.

1.4 Thesis structure

As depicted from *Figure 1.2*, the thesis is built on top of a skeleton (yellow area) containing Introduction, Background, and Conclusions. The main contributions are categorized in two macro-area (gray boxes): Application Characterization and Application Optimization.
In the following paragraphs, we detail the thesis structure per chapter:

Chapter 2: discusses the background information required to understand the thesis' contributions. We start by highlighting the main computing paradigm employed in this thesis. Then, we introduce the primary methodologies for application characterization. The chapter concludes with a thorough background of Radio-Astronomical Imaging algorithms, a real-world application with high-performance requirements we employ in the thesis.

Chapter 3: introduces the platform-independent software analysis tool for characterizing near-memory computing features. It discusses the novel memory and data parallelism metrics that we add to the original tool to enrich the application characterization. Then, we demonstrate the correlation of using these metrics with near-memory computing performance by employing the principal component analysis and near-memory computing simulation.

Chapter 4: introduces the Near-Memory computing Profiling and Offloading (NMPO) framework. NMPO is an ensemble machine learning model trained with near-memory computing simulation and hardware-dependent characterization data to predict near-memory computing suitability of unseen applications. NMPO promises to be faster than the state-of-the-art, especially in the prediction phase by employing hardware-dependent characterization, which typically has reduced overhead compared to hardware-independent techniques.

Chapter 5: presents a specific Radio-Astronomical Imaging use case. Indeed, radio-astronomers will need to process high-resolution images to detect smaller and hidden space objects in the coming years. Two-dimensional Fast Fourier Transform becomes a critical bottleneck with increased image size. In this chapter, we characterize the 2D FFT by employing hardware-dependent methodologies.

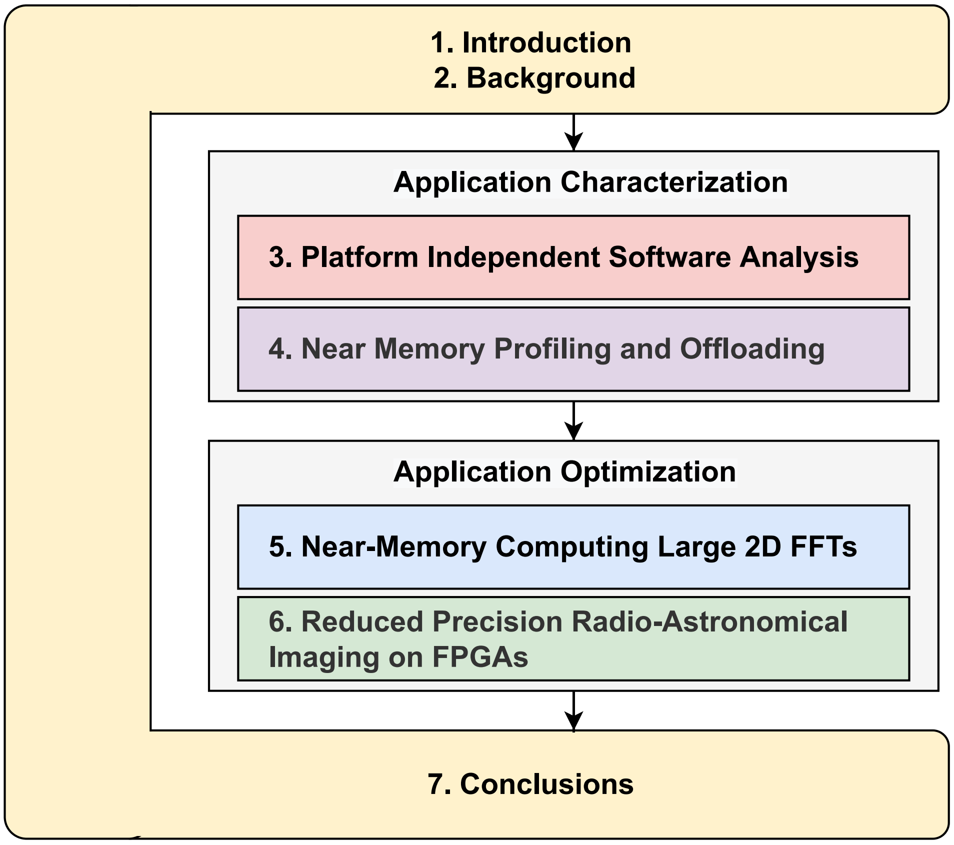

1. Introduction
2. Background
Application Characterization
3. Platform Independent Software Analysis
4. Near Memory Profiling and Offloading
Application Optimization
5. Near-Memory Computing Large 2D FFTs
6. Reduced Precision Radio-Astronomical Imaging on FPGAs
7. Conclusions

Figure 1.2: *Graphical representation of the thesis structure. The thesis develops on two macro-topics: application characterization and application optimization. Each topic has its own contributions and they are linked across the body of the thesis.*

Then, we evaluate its performance on high-performance architectures, including a Near-Memory Computing system based on the Access Processor concept and realized on an FPGA.

Chapter 6: shows an in-depth analysis by using hardware-dependent tools of one state-of-the-art radio-astronomical imager. More precisely, we focus on determining the main application bottleneck. To the best of our knowledge, this is the first time that the application tolerance to reduced-precision data types is evaluated. Then, we propose the first reduced-precision Image-Domain Gridding accelerator on FPGA. The chapter concludes with an evaluation of the prototypes presented with state-of-the-art architectures such as CPUs and GPUs.

Chapter 7: summarizes the thesis contributions and gives an overview on potential future works.

2 Background

In order to understand this thesis some background information can help the reader. This chapter starts by describing the main computing systems studied and employed for this work (*2.1*). Application characterization techniques are reported in *2.2* and this chapter concludes with describing the most significant use case of this thesis: radio-astronomical imaging (*2.3*).

2.1 High Performance Computing Systems

Modern scientific applications need High-Performance Computing in order to execute efficiently with sufficient performance. In the past decades, we have witnessed a rapid evolution in computer architecture. Especially, the CPU (Central Processing Unit), usually considered as the “heart” of a computing systems (HPC servers, laptops, smartphones, domestic devices, etc.) has deeply changed in the past years. Indeed, as shown in *Figure 2.1*, from 1970s there has been an exponential improvement in terms of frequency, number of transistors and single-thread performance. This behaviour was driven by the so-called Moore's Law [8] and Dennard's scaling Law [9]. The first one claims that the number of transistors doubles approximately every two years, while the second one states that the performance per watt double every 18 months.
However, at the beginning of the 21th century these two laws started to slow down and not be completely valid anymore [36]. Indeed, *Figure 2.1* shows that in the past two decades the number of transistor is still doubling at a lower speed, about every 18-24 months. A similar trend affects Dennard's scaling. In fact, starting

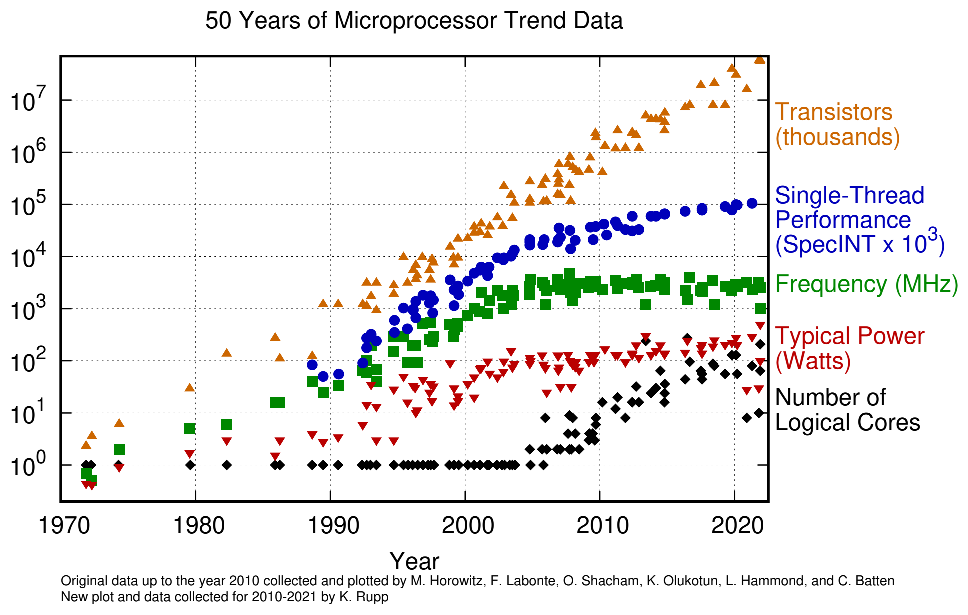
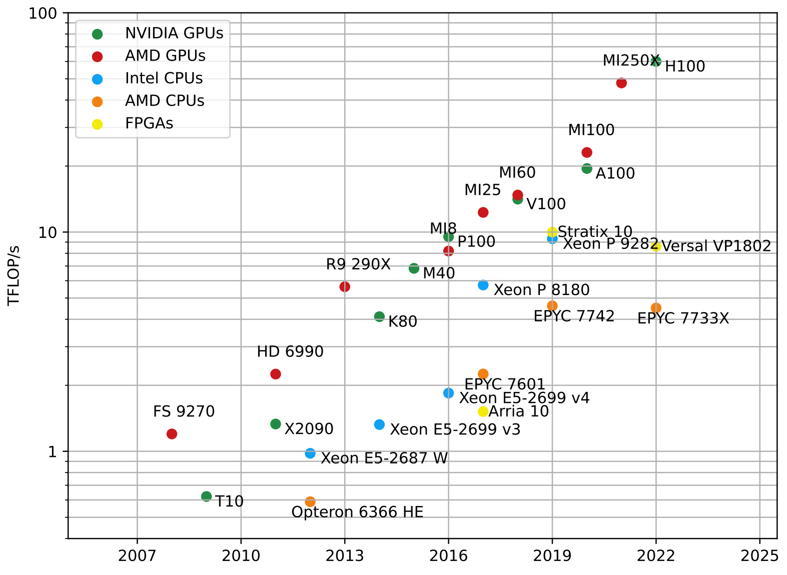


Figure 2.1: *50 years processors trend [35]: number of transistors, single-thread performance, frequency, power consumption and number of logical cores. The figure show how the number of transistor is increasing over time. This is true for the number of logical cores starting from the early 2000s. Contrariwise, the other trends are reaching a plateau.*

from the early 2000s frequency and single-thread improvement started to reach a plateau. Typical high frequency values for today's CPU are in the range of 4 to 5 GHz. However, these values can typically be achieved by a single core at a time.

Nevertheless, new solutions started to be introduced. As can be depicted from *Figure 2.1* around 2005, CPUs with multiple core started appearing on the market increasing the complexity of modern CPUs but providing higher performance for modern applications.

To accommodate the requirements of modern and performance demanding workloads, in the past two decades alternative architectures such as Graphical Processing Units (GPU) and Field-Programmable Gates Arrays (FPGAs) have been produced. In particular, these new architectures have usually higher peak performance in terms of operations per second and bytes per second at the cost of being less general purpose [37]. Indeed, these architectures work perfectly with embarrassingly parallel algorithms and typically provide better energy efficiency.



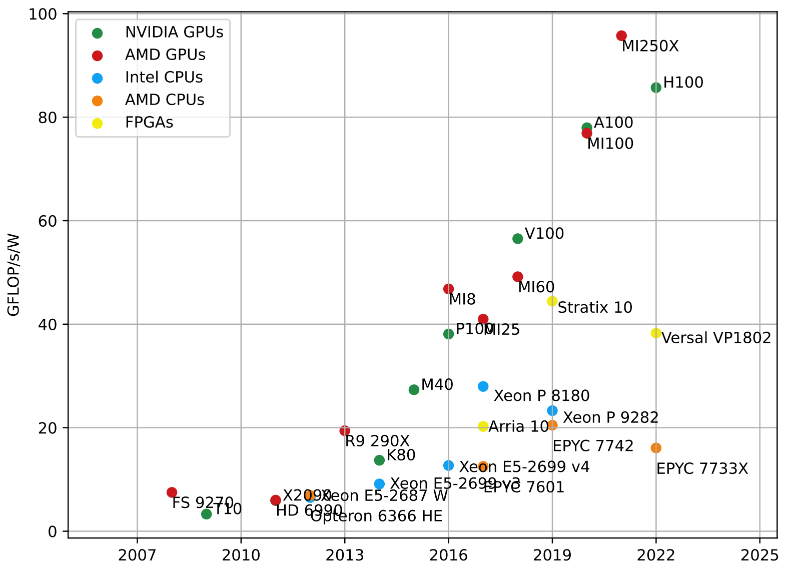


Figure 2.2: *The top figure shows the theoretical peak performance expressed in TFLOP/s for different computing architectures in the past decade. NVIDIA GPUs are reported in green and AMD GPUs in red; they show the higher values of TFLOP/s. Other architectures such as CPU and FPGAs show lower performance; they are reported in blue (Intel CPUs), orange (AMD CPUs) and yellow (FPGAs). The bottom figure presents the theoretical energy efficiency (GFLOP/s/W) of the same architectures. Usually GPUs and FPGAs are far more energy efficient than CPUs.*

For completeness, we show in *Figure 2.2* the trend over the years of the most common computing architectures (CPUs, GPUs and FPGAs) in terms of performance (TFLOP/s) and energy efficiency (GFLOP/s/W). Overall, the improvement is exponential. GPUs usually can provide the highest throughput and energy efficiency compared to the other systems.

2.1.1 Central Processing Units

A Central Processing Unit (CPU) is a key element in today's computing systems. A CPU typically consists of a small number of high-performance cores, often are in the range of 2-64 and can reach frequencies around 4-5 GHz. Usually a cache hierarchy is integrated in the CPU package. Cache memory is faster but smaller than the main memory where the application and operating system data usually resides. Cache memory temporarily stores frequently used instructions and data enabling quicker processing on the CPU cores. In most cases, the main memory uses Dynamic Random Access Memory (DRAM), which is commonly located outside the CPU package. However, this is not true for the recently released Apple M1 and M2 chips [38], where the CPU and main memory are placed in the same chip. CPU's cores usually consist of an instruction pipeline including Arithmetic Logic Units (ALU) and Registers for processing operations. Modern CPU's cores support vector operations and multi-threaded execution, therefore increasing the level of parallelism and complexity.

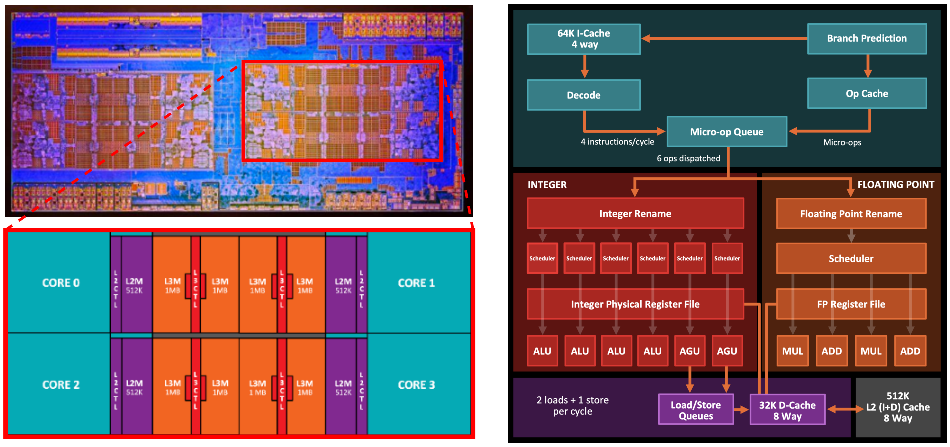


Figure 2.3: *AMD zen2 architecture. On the left, the upper figure is the layout of an 8 cores CPU, the bottom figure is a zoomed representation of a CCX section of the 8 core CPU. On the right side, the pipelined representation of a zen2 core [39].*

A layout of a relatively recent CPU is showed in *Figure 2.3.* On the left, the upper image shows the layout of 8 cores AMD CPU with zen2 architecture. Modern CPU

architectures pack a number of resources such as caches and cores in a modular fashion. The bottom image on the left side shows the layout of an AMD CCX (CPU Core Complex). It contains 4 large cores with each a L2 private cache, a shared L3 cache. Note that this may differ across vendors and architecture generations.
The right side of *Figure 2.3* depicts the structure of a modern CPU core: L1 and L2 private caches, ALUs for integer and floating-point (red and orange), pipeline queues. To maximize performance, CPUs implement pipelined designs [40] in order to increase the computation througput. The pipeline stages can be recognized in in the upper part of the core scheme: instruction decode from the instruction cache, micro-op queue, ALU execution, etc.

2.1.2 Graphical Processing Units

Graphical Processing Units (GPUs) were designed mainly for graphic tasks such as gaming and video rendering. Twenty years ago, GPUs were complicated to program for scientific computation and were mainly employed for applications with operations on image-like data structures [41]. Only with the advent of high-level programming models supporting common programming languages, such as C/C++, GPUs became popular for scientific workloads. In 2007 NVIDIA released a C-like language for GPUs, the so-called CUDA (Computed-Unified-Device-Architecture) [42]. A few years later, competitors such as AMD and Intel proposed OpenCL [41], which, compared to CUDA, is not vendor-specific.
GPU architectures have specific features that make them profitable for certain workloads compared to CPUs. GPUs usually come with fast memory soldered on the same board and are significantly faster than the CPU's main memory (called global memory). Modern GPUs are equipped with GDDR6X, or HBM2. While GDDR memory is placed on the PCB and spread around the processor, the HBM memory is located on the GPU itself and the different stacked dies communicate via microbumps and through-silicone vias (TSV). They can achieve a peak bandwidth in the order of TB/s [13]. Another key difference with CPUs is that GPU have more cores at lower frequencies: current high-end GPUs, such as the NVIDIA A100, have more than ten thousand processing elements. Each processing element has a private memory. It also includes units for floating-point (single and double precision) computations (FPU) and other units based on the model. For instance, modern NVIDIA GPUs have Special Functional Units (SFU) for transcendental operations such as sine and cosine [44]. They may also include tensor cores [45], for AI workloads, which are incredibly efficient in computing matrix multiply and accumulate. Most modern GPUs also support mixed-precision computation, including data types not commonly supported by CPUs, such as Brain Floating Point and Half Precision.
As shown in *Figure 2.4*, GPU processing elements are organized in clusters, so called Streaming Multiprocessor (SM) or Shader Engines depending on the vendor terminology. Inside an SM, the cores share an L1 memory, also called shared

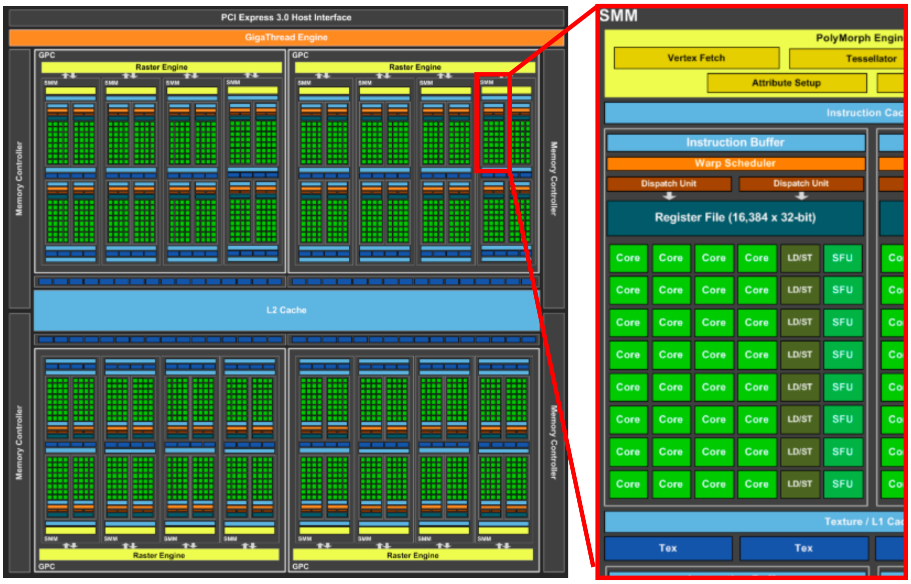


Figure 2.4: *Schematic of an NVIDIA Tesla V100 GPU [43]. On the left, the overall architecture with global memory, L2 caches and GPCs (Graphic Processing Clusters). On the right, a zoomed view of a Streaming Multiprocessor (SM). We can distinguish cores, SFUs (Special Function Units), Load/Store units, etc.*

memory. SMs are grouped in GPCs (Graphic Processing Clusters) and access a common L2 cache.
Although modern GPUs are programmable in a high-level language, their programming model is different from the ones (like C/C++) used for CPUs. GPU computation exploits many threads for parallel operations. The operations are typically defined as a Grid consisting of Thread blocks. Each thread block hosts a group of threads (warps or wavefronts) [42].

```
void vecAdd(double *a, double *b,
      double *c, int SIZE)
#pragma omp parallel for
   for(int i = 0; i < SIZE; i++){
      c[i] = a[i] + b[i];
   }
}
```

Listing 2.1: *Vector addition in C++.*

```
__global__ void vecAdd(double *a,
      double *b, double *c, int SIZE)
{
   int id = blockIdx.x*blockDim.x+
      threadIdx.x;

   if (id < n)
      c[id] = a[id] + b[id];
}
```

Listing 2.2: *Vector addition in CUDA.*

These features are explicit in the programming model. For instance, in *Listing 2.1* we show an example of vector addition on CPU, where the parallelism is inferred with an OpenMP pragma. To exploit GPUs parallelism, the code is written differently (see *Listing 2.2*). The first difference that can be noticed is that there is no for loop. The computation is split into different Thread Blocks, which a single Streaming Multiprocessor executes. Each Thread Block consists of different Threads that are offloaded to the SM's processing elements. These two dimensions (Thread Blocks and Threads) are specified by the kernel call in the host (CPU) code by using angular brackets: `vecAdd<ThreadBlocks, Threads>(a, b, c, SIZE)`.

2.1.3 Field Programmable Gate Arrays

Field-Programmable Gate Arrays (FPGAs) are integrated circuits (IC) that consist of multiple programmable block and interconnect (Tile in *Figure 2.5*) that can be programmed to execute specific computations.

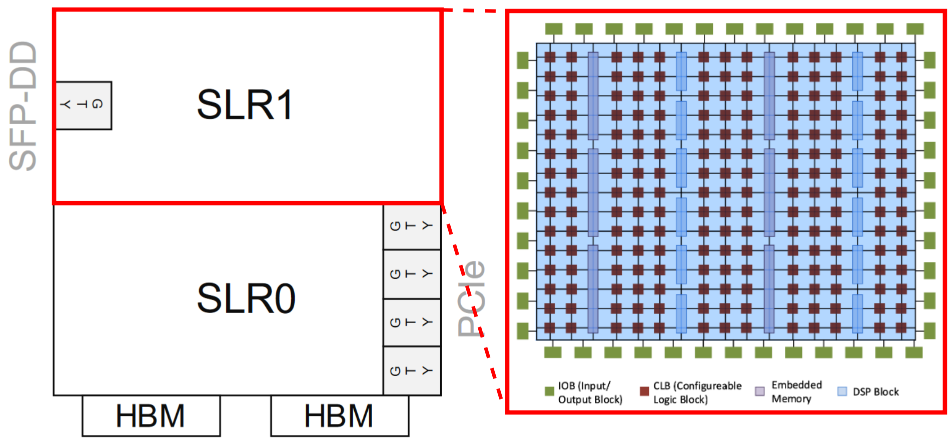


Figure 2.5: *FPGA architecture. Modern FPGAs are composed by multiple Super Logic Regions (SRL) to compete with GPUs. They also support high-speed connections such as GTY and SFP-DD.*

In the case of Xilinx FPGAs, each block contains Configurable Logic Blocks (CLB) with LUTs (lookup tables) and FFs (flip flops) to generate logic circuits and small memories. Modern FPGAs also have DSP (digital signal processing) units, specially optimized for multiply and accumulate operations, and on-chip memories (BRAM, URAM, etc.) for storing high-re-used data or buffering inputs/outputs. The blocks mentioned above are programmed and connected to form digital functions. With the recent advancement in FPGA technology, they can easily perform numeric computations in floating-point and customizable/arbitrary precision arithmetic. This makes FPGAs appealing over architectures like CPUs and GPUs that support a set of pre-defined data types. FPGAs were programmed in the past

with relatively low-level Hardware Description Languages (HDLs), such as Verilog and VHDL. To avoid this time-expensive and tedious work, FPGAs vendors such as Xilinx and Intel support High-Level Synthesis Tools allowing programming FPGAs with High-Level Languages such as C++ and python. Moreover, vendors make available high-level APIs (Application Program Interface), which usually are usually based on OpenCL, to easily program and offload computation on the FPGAs.
However, the hardware generation (implementation), which provides the configuration, called bitstream, still requires several hours. Another important feature of FPGA devices is re-configurability. More precisely, it is possible to re-configure the programmable logic on at the run-time to support different computations or adapt the hardware to dynamic requirements. These features make FPGAs attractive candidates for High-Performance Computing and possible competitors with GPUs.

2.1.4 Near Memory Computing

Differently from traditional computing architectures such as CPUs, near-memory computing (NMC) processes data near the main memory, in certain cases near storage memory, thereby avoiding the classical data movement throughout the cache hierarchy, which usually causes memory bottlenecks and increase the power consumption. First NMC ideas date back to 1970 [46]. However, given the hardware limitations at that time, the first prototypes started appearing not before early 1990s [47–49]. Vector IRAM (VIRAM) [50] is an example. Researchers developed a vector processor with an on-chip embedded DRAM (eDRAM) to exploit data parallelism in multimedia applications. This solution did not reach the market, although the results were quite promising.
Recently, two key factors make NMC regain attention: the rising number of data-intensive applications and the memory technology improvement (see *Figure 2.6*). More precisely, we are continuously witnessing an unstoppable increase of data requirements for applications in diverse fields such as health, social media, radio astronomy, etc. [51].
Three-dimensional stacked memory is one of the most important reasons of the NMC regained visibility. This technology can be used in 2.5D and 3D stacking configuration with the logic layer. In *Figure 2.7* we show an example of 3D stacking where memory layers and logic layers communicate with through-silicon via's (TSVs) reducing memory access latency, power consumption and enable high memory bandwidth [52]. In the case of 2.5D, mostly used in commercial solution (GPUs, FPGAs), the 3D-stacked memory communicate with the logic layer with an interposer, thus not being stacked. Example of 3D-stacked memory technologies are HBM (High Bandwidth Memory) [11] and HMC (Hybrid Memory Cube) [10].
Near-memory computing approaches have been studied and categorized by Singh et al. [14]. Briefly, NMC systems can be very different. The main parameter

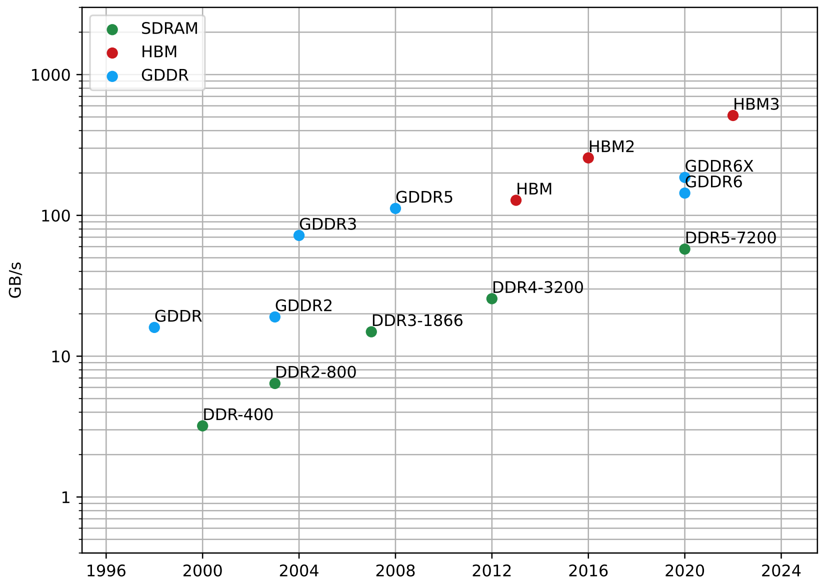


Figure 2.6: *Memory bandwidth trend of the most employed memory technology in HPC. In green SDRAM, typically used for CPUs; in blue GDDR, which is employed in GPUs. HBM is in red; it is used in high-end GPUs and FPGAs, but also in novel architectures such as NMC.*

that varies in NMC systems is the main memory: HBM [53, 54], HMC [21] storage. Furthermore, there is an heterogeneous landscape in the state-of-the-art of processing approaches: simple cores [21], fixed functions [55], etc.
Moreover, since producing near-memory computing systems is costly, most of the research validation experiments are carried out with simulators [21, 56–58].

2.2 Application characterization

Application characterization typically refers to the methodologies employed to collect performance-critical metrics in current and emerging applications. This is a challenging and crucial task, especially with continuously evolving hardware and software. Indeed, application characterization tools are fundamental when co-designing the hardware and software for modern workloads.

2.2.1 Goals

Based on the characterization methodology, the achievable goals could be different. We summarize the most relevant ones as follows:

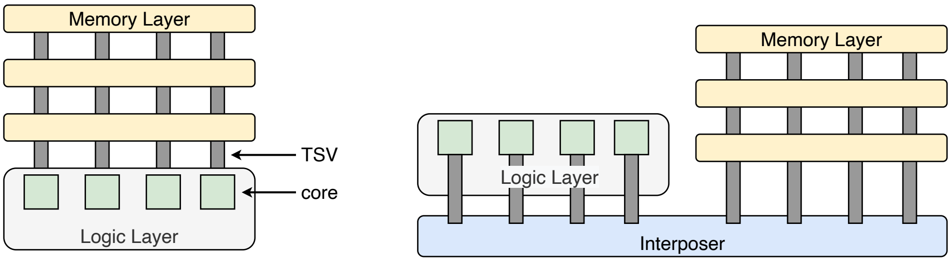


Figure 2.7: *On the left an example of Near-Memory Computing (NMC) architecture with a 3D-stacked chip. The 3D-stacked memory is connected to the compute units or cores, placed on the logic layer, with through-silicon via's (TSVs). On the right, an example of 2.5D NMC architecture: the logica layer is connected to the memory layer with through-silicon via's (TSVs) through a layer called interposer.*

Application hot spots: according to Amdahl's Law [8], reported in *Equation 2.1*, the speedup strictly depends on the critical hot spot that can be parallelized or accelerated. Therefore, finding application hot spots guarantee to optimize those application portions that will return a performance improvement:

$$S_{latency}(s) = \frac{1}{(1-p)+\frac{p}{s}} \tag{2.1}$$

where p is the application portion that can be parallelized and s is the section that must run sequentially.
This is usually achieved by individuating workload's critical sections that are frequently repeated over time, and that can be parallelized. First attempts date back late 1960's when IBM developed a monitoring device to profile CPU execution [59]. In the past 60 years, a large number of tools have been developed and proposed for this purpose, such as callgraph analysis tools [60,61] or hardware proprietary tools [62–64].

Hardware bottleneck: it is crucial to understand where the application stalls on specific hardware architecture. Indeed, new hardware generation may introduce a difference in their design that may impact the performance of pre-existing software. This is crucial to understand the current hardware's limitations and decide to either change architecture or, if possible, adapt the software to match it. Many tools capable of extracting hardware counters, such as Intel VTune [64], NVIDIA Nsight Compute [62], are available nowadays. Often these tools integrate the roofline model or can output the necessary metric to draw it. The roofline model is a widely employed performance model to assess achieved performance and hardware bottleneck [65].

The roofline model (see *Figure 2.8*) can be drawn for almost all the computing architectures such as CPUs, GPUs, and FPGAs. The key information that is needed for a basic roofline model are the peak compute and memory roofs. These are typically computed theoretically or empirically [66]. In the example, in *Figure 2.8*, the horizontal line represents the peak performance of the considered architecture, and it is usually expressed in floating-point operations per second (FLOP/s). The diagonal line represents the peak memory bandwidth.

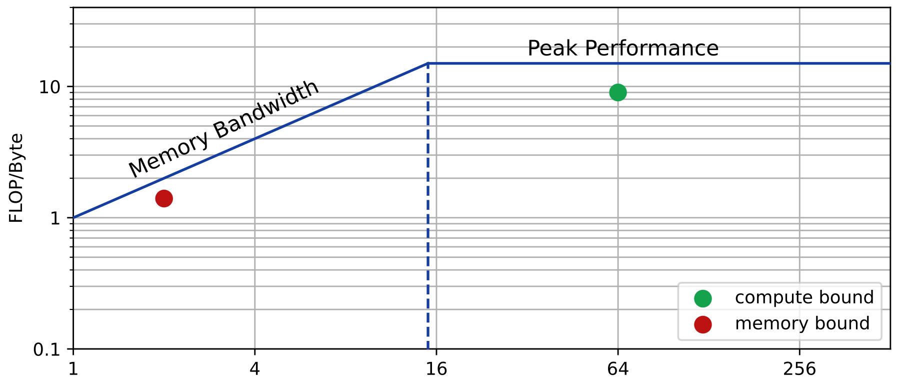


Figure 2.8: *Example of roofline model. The blue lines represent the architecture limits in terms of peak performance and memory bandwidth. The roofline model is helpful in distinguishing memory and compute-bound applications (see the green and red dots).*

The roofline model shows on the y-axis the application achieved performance and on the x-axis the arithmetic intensity (FLOP/Byte). Arithmetic intensity is usually the factor that makes an application memory or compute bound. Indeed, if the application is underneath the compute roofs, it is compute bound, otherwise it is memory bound.
Application intrinsic analysis: the design-space exploration of emerging and future systems necessitates application characterization in terms of intrinsic application features. Indeed, it is crucial to understand application-related characteristics such as Instruction-Level Parallelism, Memory Entropy, and Task-Level Parallelism to efficiently design the system mentioned above. The information extracted is usually employed to build an architectural model or performance model to predict application performance on unavailable or to be designed computing systems [67].

Adaptiveness: Modern compiler such as LLVM Clang [68] embed a large variety of analysis tools in their toolchain to be able to identify bottlenecks such as race conditions [69] or detecting particular code patterns [24,70] that can be optimized by the compiler itself. Run-time managers represent another example. They are typically software routines that run parallel with applications on a system and employ application profiling to determine stalls or offloading opportunities. This is usually done by analyzing the application performance, and hardware utilization to adapt on-the-fly the application offloading, e.g., by dynamically adapting the task scheduling [71, 72].

2.2.2 Taxonomy

The numerous profiling and characterization tools that have been proposed over the years can be distinguished by their features.

- **Hardware dependency:** the profiling may only include the code analysis (hardware independent) or can in addition take into consideration hardware features (hardware dependent).
- **Metrics**: collected metrics can be really different among characterization tools. Common metrics are execution time, floating-point operation per second (FLOP/s), arithmetic intensity (FLOP/Byte), energy efficiency, Instruction-Level-Parallelism (ILP), etc.
- **Application stage:** analysis can be done at different application stages: source code, compilation time, and run-time.
- **Granularity:** profiling tools can work at different granularity starting from coarse grain such as event-based profilers to fine-grained such as profiling at instruction level.

Applications can be analyzed with different approaches. Often a mixture of methodologies can be applied:

- **Event-based:** this technique triggers the analysis based on software events such as function calls, class load or unload, etc [64].
- **Statistical:** the profiler collects information by sampling data. Perf, a profiler utility available in the Linux kernel, [73] or PAPI (Performance Application Profiling Interface) are able to collect hardware counter measurements at a certain sampling frequency.
- **Instrumentation:** the aim of this method is to inject additional instructions to collect application information. This technique usually reduces the applications performance. Instrumentation can be done in different ways: 1) the user can manually add instructions to the application code, e.g., PAPI [74]; 2) a tool can automatically instrument the source code [75] or the compiler intermediate representation; 3) other approaches perform the instrumentation at binary level [76] or at run-time.

- **Simulation:** the application dynamically runs under an instruction set simulator (ISS), a simulation model capable of reproducing an architecture behavior. This is typically coded in high-level languages such as C++. Simulators can model entire systems or specific components. For instance, Ramulator [77] and DRAMSim3 [78] are cycle-accurate memory simulators. Gem5 [79] and Zsim [80] are employed to emulate the behaviour of CPUs, while GPGPUsim [81] models GPUs execution.

- **Analytical models**: are typically quantitative and less precise, compared to, e.g., simulator, estimation based on a set of mathematical equations. They quickly estimate the application performance based on collected application properties and hardware features. In this context, e.g., Gysi et al. [82] propose a fast analytical model of fully associative caches, based on static analysis using the LLVM compiler framework, that outperforms cache simulators by several order of magnitude with an error below 0.6%.

2.2.3 State of the art

In this thesis, we mainly focus on the difference between hardware-independent and hardware-dependent analysis tools. Moreover, while the first ones can be more accurate and helpful for unavailable hardware, the second ones are easier to use and quicker. We summarize the most relevant state-of-the-art approaches and tools in the following paragraphs.

Hardware independent analysis A hardware-independent analysis can be done both statically and dynamically. For instance, Jordans et al. [83] analyzed the processor parallelism using a static analysis of the LLVM's IR, which is the Intermediate Representation employed by the LLVM compiler. This work aimed to estimate the level of parallelism in a workload and propose optimization strategies focused on VLIW architectures. Another static approach was proposed by Eusse et al. [84]. They used the LLVM framework to perform a pre-architectural performance estimation coupling high-level synthesis (HLS) to shorten design times.
On the other hand, dynamic analysis can collect the difference when changing the workload's problem size. Cabezas [85] proposed a tool that can extract different features from workloads but has many limitations: the compiler community no longer supports the used LLVM interpreter, and the target applications should be single threaded. Another tool has been developed by Shao et al. [22]. It can extract interesting metrics such as memory entropy and branch entropy. However, this tool has some limitations: it is based on the IDJIT IR (just-in-time compilation) that has compatibility problems with OpenMP and MPI, thus being limited to sequential applications. Vector Fabrics [86] developed several commercial LLVM-based tools. For instance, based on dynamic analysis, Vector Fabrics' Pareon Profile tool was used to explore opportunities and bottlenecks for

parallel execution of $C/C++$ code; Vector Fabrics' Pareon Verify, which uses dynamic analysis to find bugs in $C/C++$ application code.
One of the most recent hardware agnostic tools is PISA (Platform-Independent Software Analysis) for workload characterization was presented by Anghel et al. [23]. PISA can analyze multi-threaded applications supporting the OpenMP and the MPI standards.

Hardware dependent analysis As mentioned above, hardware-dependent analysis dates back to the late 1960s [59]. Over the years, additional tools have been researched. For instance Tjaden et al. [87] propose instruction-level parallelism (ILP) estimation. They took into account: memory accesses, data registers, operand availability, and procedural dependency. Later, Theobald et al. [88] improved the ILP estimation by analyzing instruction traces and including the effects of various memory reuse policies and long-latency operations.
Then, in 1982 Graham et al. [89] introduced *gprof*, an improvement of the existing Unix-prof tool. Its novelty was the utilization of a call graph to analyze both the time spent on methods and their impact. *Gprof* generated an overhead of about 30% compared to the normal execution.
Hoste et al. [90,91] developed a tool based on ATOM/Pin [92]. This tool performs code instrumentation based on which it can extract ISA-dependent metrics.
Caparros et al. [93] proposed a tool that profiles the binary execution dynamically. They estimated: instruction-level parallelism, thread-level parallelism, potential data-level parallelism, and inter-task data movement. However, this tool was limited by the influence of the architecture characteristics, and they could analyze only single-thread code.
Another work was done by Ferdman et al. [94], using x86 architectures, they tried to analyze a benchmark suite for the cloud. They showed, for certain applications, an over-provisioning of the memory bandwidth and low memory-level parallelism. However, they used hardware performance counters, and this limits the analysis to the employed hardware, especially for micro-architecture features such as cache size, issue width, etc.
Nowadays, a large number of commercial target-dependent tools are available. One of the most common ones [95] is included in Linux OS, and it is called *perf*. Using the available performance counters *perf* can extract interesting information, e.g., cache miss, rate and CPU utilization. The *nvprof* (*NVIDIA*) profiling tool [62] is specialized on GPU workload analysis. It can extract a timeline of CUDA-related activities on both CPU and GPU, including kernel execution, memory transfers, CUDA API calls, and events, or metrics for CUDA kernels.
Also, Intel has developed some interesting analysis tools; such as 1) *Intel VTune Amplifier* [64] that can extract a wide range of analysis metrics about CPU, Memory, and GPU; this is based on the work of Yasin et al. [96]. It is easy to use, and the overhead added to the application is very low; 2) *Intel Advisor Roofline* [97] that can plot a roofline model of the analyzed application allowing

memory and compute boundedness analysis.

2.3 Radio-astronomical imaging

An important use case is represented by radio-astronomical imaging. Indeed, applications of this field are fundamental to answer important questions such as: how do galaxies and planets form? are we alone in the universe?

These problems have high-performance requirements. For instance, the Square Kilometre Array (SKA) radio-telescope will be the largest radio-telescope in the world hosted in Australia and South Africa. The first construction phase of SKA will finish before the 2030. The processing facilities that the SKA consortium is planning to build for the low and mid frequencies consist of large Science Data Processors. Each of them has a peak performance in the order of 6.50 PFLOP/s, and a thermal design power in the order of 125 MW [4].
We introduce interferometry and imaging in *Section 2.3.1*. Then we highlight some details about the Image Domain Gridding (see *Section 2.3.2* and Deconvolution algorithms (see *Section 2.3.3*.

2.3.1 Interferometry and Imaging

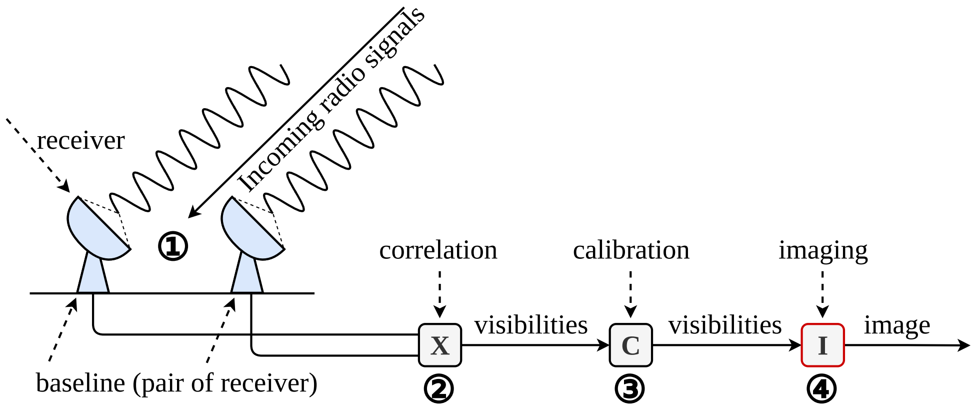


Figure 2.9: *Radio astronomy image acquisition: the incoming radio signals are digitized and then correlated and calibrated before the imaging step is executed. We focus on the imaging step, which is highlighted in red.*

A radio telescope detects electromagnetic waves that originate from radio sources in the universe. The signals are used, among other things, to construct a map of the sky containing the positions, intensity, and polarization of the sources.

Radio telescopes such as LOFAR (Low Frequency ARray) [98] and SKA1-Low [5] are comprised of many (small) dipole antennas that measure two orthogonal polarizations of the radio sources, while other radio telescopes, such as the VLA [99], MeerKAT [100] and SKA1-Mid [101], are based on an array of dishes. As shown in *Figure 2.9*, a station consists of multiple antennas, for which the signals for every distinct frequency channel are combined. The signals of a pair of stations (called a baseline) are multiplied and integrated (correlation ②) for a short period of time (in the order of seconds), thus producing a single visibility (a 2x2 matrix). The data that the telescope produces (the `visibilities`) is, therefore, a three-dimensional matrix (with indices number of baselines, frequency channels and correlations). The relation between visibilities and sky brightness is given by a measurement equation; see [102] for complete details.
The visibilities are first calibrated (③) and next used to reconstruct the sky brightness in the observation direction using an imaging step (④) [103].
This thesis mainly focuses on ④. The imaging step (see *Figure 2.10*) starts with an empty sky model and it consists of an iterative process: 1) the `inversion` step is used to produce a `dirty image`; 2) one or more bright sources are detected in this image by a deconvolution algorithm such as `CLEAN` (see also Section 2.3.3); 3) a `model image` is created, which contains all of the sources in the sky model; 4) visibilities corresponding to this model image are `predicted`; 5) subtracting the predicted visibilities from the measured (and calibrated) visibilities yields `residual visibilities`. This process subtracts strong sources from the measurements, which mask the more interesting weak sources. This step is repeated until the sky model converges. Finally, the sky model is used to create the sky image.
The inversion and prediction steps comprise of 2D FFT and a `gridding` or `degridding` step. The gridding and degridding steps are typically the most compute-intensive image processing steps. To attain high-quality sky images, they need to correct for Direction-Independent Effects (the curvature of the earth, `W-Term` correction) and Direction-Dependent Effects (such as ionospheric effects, `A-Term` correction). The W-Term can be corrected by applying a convolution kernel to every visibility. The required convolution kernel could be huge, depending on parameters such as the field-of-view and distance between receivers. A-Term correction requires these convolution kernels to be different for every receiver and change over time according to changes in the Direction-Independent effects. These properties make imaging with correction for W-Terms and A-Terms particularly challenging.

2.3.2 Image-Domain Gridding

Image-Domaing Gridding (IDG) is a state-of-the-art algorithm for both gridding and degridding [105]. IDG performs both W-correction and A-correction in the image domain, avoiding large convolutions functions. The algorithm performs gridding and degridding using `subgrids`, which represent low-resolution sky im-

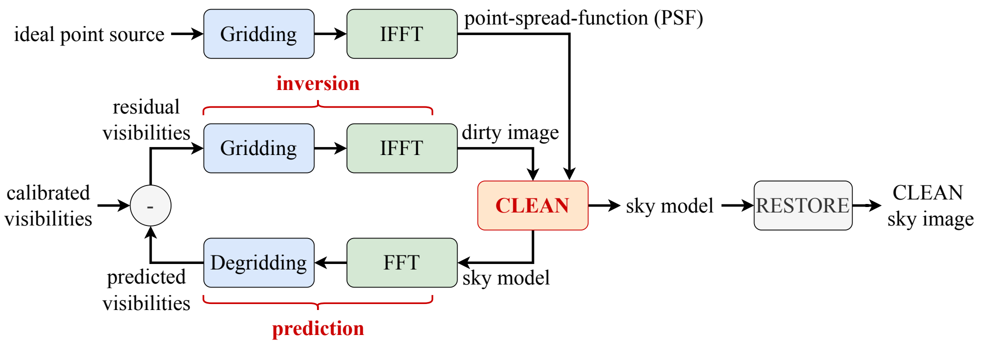


Figure 2.10: *High-level schematic of the radio-astronomical imaging step. The three main phases inversion, prediction and deconvolution (or CLEAN), are highlighted in red. The critical computations are typically the inversion and the prediction steps, which include Gridding/Degridding and 2D Fast Fourier Transforms. We included the point-spread-function (PSF) computation, which is an inversion step with an all-ones matrix of visibilities and it is used during the CLEAN.*

ages for a subset of visibilities. This approach exposes a lot of parallelism (subgrids can be processed in parallel), which makes it highly efficient on parallel hardware such as GPUs [4]. In IDG, gridding comprises three steps: 1) visibilities are gridded onto subgrids; 2) subgrids are Fourier transformed; 3) subgrids are added to the larger final grid. IDG degridding comprises these steps in reverse order. Refer to [4, 105] for all the details on this algorithm and a formal derivation.
Image-Domain Gridding performs much better [106] than classical gridding/degridding algorithms, such as W-projection [107] or AW-projection [108]. It also employs W-terms to solve artifacts around sources away from the phase center in wide-field imaging. Moreover, IDG image quality is higher than W-projection because IDG, like AW-projection, corrects for DDEs (direction-dependent effects, also called the A-terms), but the computational costs for such DDE corrections are much lower for IDG than for AW-projection [4]. IDG also has higher per-visibility accuracy compared to the other algorithms [105].

2.3.3 Deconvolution

The objective of a CLEAN (or deconvolution) algorithm is to detect sky sources by iteratively finding the brightest peaks in a dirty image and fitting a sky model. In *Figure 2.12* we show an example of a dirty image and the corresponding image after a deconvolution algorithm has been applied. The CLEAN image shows lower noise compared to the dirty image.
Many CLEAN algorithms have been proposed in the literature. We briefly de-

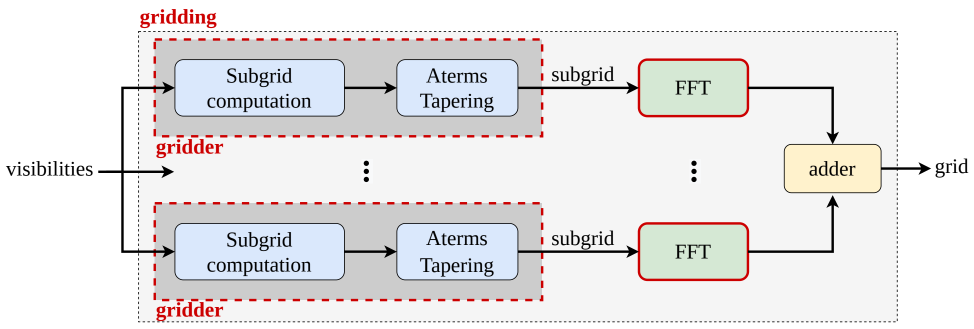


Figure 2.11: *Gridding high-level representation. It consists of multiple gridder computations (subgrid computation and tapering) and FFTs that process the input visibilities into subgrids. Then the subgrids are processed by the adder to obtain a grid. The gridder and FFT, red boxes, are the focus of this and the related work [104].*

scribe the commonly used CLEAN algorithms in radio-astronomical imaging:

Högbom: it is the simplest CLEAN algorithm. After the dirty image is generated from the imaging step (gridding and IFFT), the Högbom CLEAN tries to remove the noise in the image. This is done iteratively, looking for the maximum value in the image. Then, the algorithm subtracts the Point Spread Function (PSF), which is a function dependent on the telescope used. It is computed like the dirty image using as input a Visibility array with values equal to 1, multiplied by a gain factor. After a certain number of iterations or when a certain threshold (e.g., 3σ of the standard deviation) is reached the algorithm stops. This algorithm does not include the prediction step.

Clark [109]: it is an improvement of the previously described algorithm, adds a feedback loop and tries to remove alias errors. It is possible to distinguish between `major` and `minor iterations` in this case. The minor iterations are represented by the peak search, similar to Högbom (the CLEAN box in *Figure 2.10*). Then, the model image is Fourier transformed and subtracted from the dirty image. This is the so-called major iteration.

Cotton-Schwab [110] **and Multiscale** [111, 112]: are the most employed and modern CLEAN algorithms, they have in common the prediction phase, thus including the degridding algorithm. Here the subtraction is done at the visibilities level reducing the pixelation error. More precisely, for these algorithms, a major iteration consists of an entire iteration to transform the data from the visibility domain to the image domain (inversion). The major iterations are executed until a threshold is reached, e.g., until 80% of the flux (which is a power density measure)

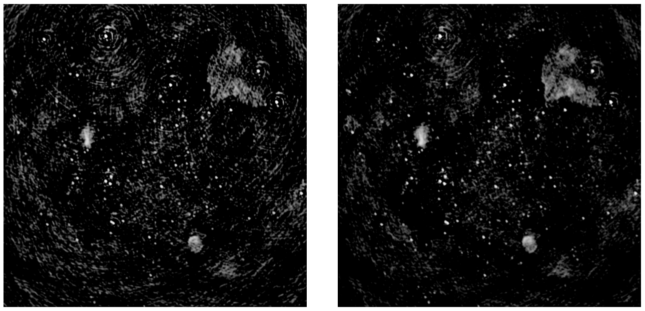


Figure 2.12: *Comparison between dirty image (left) and CLEAN image (right) applying Cotton-Schwab algorithm. The CLEAN image has reduced noise, e.g. rotation lines around the image(visible especially at the corners).*

is removed from the dirty image during the minor iterations. The Multiscale algorithm operates on a set of residual images obtained by convolving the dirty image with different scale sizes. The peak subtraction step is performed on all the scaled imaged, and only the subtracted components are stored in the CLEAN component table. After being scaled, positioned, and convolved, the final image is obtained by adding the components. It decreases the effects of the pedestal of uncleaned flux and strong sidelobes present in the dirty beam (or Point-Spread Function), which are referred to as "clean bow", around bright resolved structures and has better convergence properties [113].
We employ in our analysis work, presented in *Chapter 6*, the Cotton-Schwab algorithm, which is available in the state-of-the-art imager WSClean [114].

3 Platform-Independent Software Analysis

Application characterization is typically employed to determinate application bottlenecks and understand application features to select which kernel must be optimized and the most suitable optimization or hardware. Thus, profiling applications is a critical task to reach the goal of high performance and high energy efficiency. Emerging computing architectures such as near-memory computing (NMC) promise high performance for specific applications by reducing the data movement between CPU and memory. However, detecting such applications is not a trivial task.
This chapter presents an extension of the Platform-Independent Software Analysis (PISA) tool based on the LLVM framwork (see *Section 3.1*) with NMC related metrics such as memory entropy, spatial locality, data-level, and basic-block-level parallelism that are described in *Section 3.2.* By profiling a set of representative applications and correlating the metrics with the application's performance on a simulated NMC system, we verify the importance of those metrics. We demonstrate which metrics help identify applications suitable for NMC architectures (see *Section 3.3*). Finally, we present in *Section 3.4* the related work and in *Section 3.5* the conclusions.

This chapter is based on: S. Corda et al., "Memory and Parallelism Analysis Using a Platform-Independent Approach", SCOPES 2019 and S.Corda et al., "Platform Independent Software Analysis for Near Memory Computing", DSD 2019

3.1 Platform-Independent Software Analysis

Especially for emerging technologies like near-memory computing it is crucial to understand the intrisic behavior of applications. This can be done with hardware-agnostic profiling tools. Therefore, we here we present the LLVM framework in *Section 3.1.1* and based on top of it PISA, the Platform-Independent Software Analysis tool (*Section 3.1.2*).

3.1.1 LLVM framework

The LLVM (Low Level Virtual machine) compiler framework is a collection of tools, libraries, and header files to perform transparent, lifelong program analysis and transformations for arbitrary software. LLVM mainly consists into two parts: a code representation with helfpul features for analysis and transformation; and a compiler design that makes use of the mentioned above representation [68].
LLVM Intermediate Representation The LLVM core uses an intermediate representation (IR) to represent the application code in a generic way. LLVM's IR is generated through a front-end from the application source code. Then, it can be used to perform analyses or optimizations using the *opt* tool. LLVM's IR can be executed through an interpreter or can be translated into a target-specific executable binary. Fundamental properties of LLVM's IR are: 1) it is independent of the source language and the target architecture; 2) it is a low-level and RISC-like instruction set allowing to have an unlimited number of virtual registers in static single assignment (SSA) form. These features are essential to simulate an ideal machine [40] that consists of a Von-Neumann system with unlimited resources, such as cores, registers, or in other words when the goal is a perfect machine without the traditional hardware limitations. LLVM's IR has a hierarchical structure: a *module* that represents the application and contains functions and global variables; a *function* that is a set of basic-blocks; a *basic-block* that consists of *instructions* and represents a single entry and single exit section of code.
Compiler framework LLVM also includes a compiler framework, which provides its own compiler called `clang` and some utilities. More precisely, this compiler allows sophisticated optimization at different stages, e.g., runtime or idle, because it preserves the application's LLVM representation during its lifetime. Moreover, among other things, the compiler can generate efficient binaries offline and support custom user-based profiling optimization at different stages [68].

3.1.2 PISA Tool

PISA's architecture is shown in *Figure 3.1*. Initially, application's source code, e.g. C/C++ code, is translated into the LLVM's IR. Then, *mem2reg* optimization is used to obtain the SSA format. This is done by removing *alloca* (instruction that allocates memory on the stack frame of the currently executing function), instructions for non-aliased scalar variables. In this way, these variables are put in virtual registers. PISA exploits the *opt* tool to perform LLVM's IR optimizations and to perform the instrumentation process using an LLVM pass. This process is done by inserting calls to the external analysis library throughout the application's IR. The last step consists of a linking process that generates a native executable. On running this executable, we can obtain analysis results for specified metrics in JSON format. PISA can extract metrics such as instruction-level parallelism, data reuse distance, and instruction mix.

The analysis reconstructs and analyzes the program's instruction flow. This is possible because the analysis library is aware of the single entry and exit point for each basic-block. All the instructions contained in the basic-block are analyzed using the external library methods. Only when a load/store or a call is encountered the execution is interrupted in order to allow a dynamic analysis. In case of load/store, the analysis is restarted by an instrumentation function after the load/store instruction. In case of a call, the current state of the basic-block is saved and then PISA continues analyzing the basic-blocks pointed by the call.

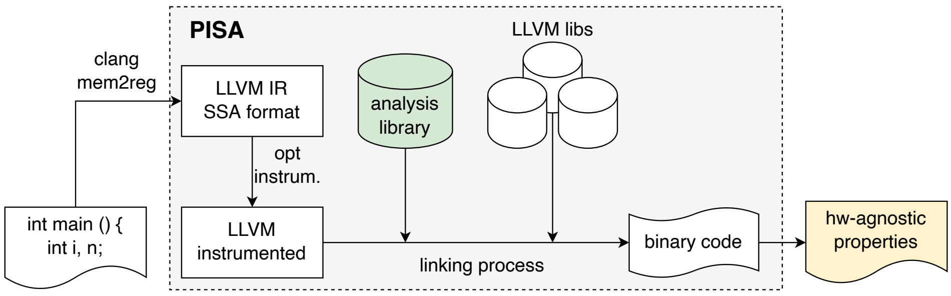


Figure 3.1: *Overview of the Platform-Independent Software Analysis Tool (gray box). The application code is converted into LLVM IR format and then instrumented with the opt tool. Then, it is linked with the custom PISA and the LLVM libraries into a hardware-agnostic binary code. This binary code can be run to obtain hardware-independent application properties. The thesis contributes in extending the analysis library (green box) to extract parallelism and memory metrics (yellow box).*

A useful feature of PISA is the support of the MPI and OpenMP standards allowing the analysis of multi-threaded and multi-process applications. In multi-threaded applications, PISA analyzes each thread individually, allocating separate data structures for each of them and distinguishing them by the thread ID because the address space is shared. For multi-process applications, this is not needed because each process has its private space for data. The overhead of this tool depends on the analysis performed. On average the execution-time of the instrumented code is increased by two to three orders of magnitude in comparison to the noninstrumented code. On the other hand, since the analysis is target-independent, this has to be performed only once per application and dataset.

Table 3.1: *Main PISA's characterization metrics with description.*

Metric	**Description**
Instruction Mix	Instruction count for each instruction category
Instruction-level parallelism	Average number of instructions that be can run in parallel
Branch entropy	The measure of randomness of branch outcomes
Data temporal reuse	Number of unique memory accesses between two accesses to the same location
MPI communications	Data volume exchanged by MPI pairs

3.2 Metrics

Memory and parallelism metrics are essential to characterize applications for near-memory computing systems. In this section we present the metrics we integrated into PISA. We focus on the memory behaviour, which is essential to decide if an application should be accelerated with a NMC architecture, and on the parallelism behaviour, which is crucial to decide if a specific parallel architecture should be integrated into an NMC system.

3.2.1 Memory entropy

The first metric related to memory access behavior that we added is the memory entropy. The memory entropy measures the randomness of the memory addresses accessed. If the memory entropy is high, which typically lead to a higher cache miss ratio, the application may benefit from 3D-stacked memory because of the volume of data moved from the main memory to the caches. In information theory, Shannon's formula [115] is used to capture entropy. We embed in PISA, the formula defined by Yen et al. [116] that applies Shannon's definition to memory addresses:

$$\text{Memory\_entropy} = -\sum_{i=1}^{2^n} \hat{p}(x_i) log_2 \hat{p}(x_i) \tag{3.1}$$

where x_i is a n-bit random variable, $\hat{p}(x_i)$ is the occurrence probability for the value x_i and 2^n is the number of values that x_i can take. $\hat{p}(x_i)$ is defined by:

$$\hat{p}(x_i) = \frac{1}{d} \sum_{j=1}^{d} I(a_j = x_i) \tag{3.2}$$

where:

$$I(a_j = x_i) = 1\ if\ (a_j = x_i),\ 0 \text{ otherwise, and } 0log0 = 0 \tag{3.3}$$

In the last formula the addresses are represented as $\{a_j\}_{j=1}^{d}$, where d is the number of different addresses accessed during the execution. Each address is in the range $[0, 2^{n-1}]$, where n is the length of the address in bits. If every possible address has the same occurrence probability the entropy is n; if only one address is accessed the entropy is 0. Otherwise the entropy is within 0 and n. The memory entropy metric does not distinguish whether the accesses contain sequential patterns or random accesses. Therefore we need additional metrics, like spatial locality.

3.2.2 Data reuse distance for multiple cache-line size and spatial locality

Data reuse distance or data temporal reuse (DTR) is a helpful metric to detect cache inefficiencies. The DTR of an address is the number of unique addresses accessed since the last reference of the requested data. This metric is present in the default framework. However, the tool could compute it only for a fixed cache line size, which represents the address granularity. We extend the DTR computation and compute it starting from the word size, such as 32 bit for single-precision floating point, to the value selected by the user. This extends the available analysis opportunities e.g. we use it to compute the spatial locality metric.
Spatial locality, which measures the probability of accessing nearby memory locations, can be derived from DTR. We extend PISA with the spatial locality score inspired by Gu et al. [117]. The key idea behind this spatial locality score is to detect a reduction in DTR when doubling the cache line size. To estimate the spatial locality in a program two elements are fundamental: 1) histograms of data reuse distance for different cache line sizes, 2) distribution maps to keep track of changes in DTR for each access doubling the cache line size.

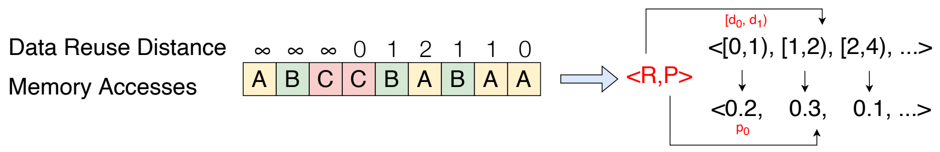


Figure 3.2: *Data Reuse Distance histogram illustration: on the left part, we show memory accesses (A, B, and C are addresses), and on top of them, we highlight the Data Reuse Distance values. In the beginning, we have cold misses represented by an infinity symbol. On the right, we have the Data Reuse Distance histogram definition with the pair <R, P>, where R is a set of bin ranges, and P is a set of probabilities.*

Histograms are used to compute the DTR distribution probability for different cache-line sizes. In [117] the reuse signature has been defined as a pair $< R, P >$ (see *Figure 3.2*), where R is a series of consecutive DTR ranges of bins, represented as: $r_i = [d_i, d_{i+1})$. These bins are a logarithmic progression defined as: $d_{i+1} = 2d_i (i \geq 0)$. P is the distribution probabilities p_i of the bin r_i. This reuse signature is used later to normalize the results.
The next step consists of building a distribution map. This map keeps track of each change in the DTR for every access. The distribution map has i rows representing the bins using a cache line size b and j columns representing the bins using a doubled cache line size $2b$. Each cell is the probability p_{ij} of the bin i using a cache line size b to change in a bin j using a cache line size $2b$. Differently from [117] we compute the sum of the cells in a row where $i < j$. as shown in *Equation 3.4*. We do that because we want to express all the changes in data reuse distance.

$$SLQ(i) = \sum_{j=0,\ \ldots,\ j<i} p_{ij} \tag{3.4}$$

To compute the spatial locality score related to a pair of cache line sizes $< b, 2b >$ we first compute the absolute values of the weighted sum that uses the probabilities p_i included in the reuse signature and then use the formula proposed by [117] to calculate the total score, which is the logarithmic weighted sum of absolute values:

$$SLQ = \frac{\sum_{\text{all } b} | \sum_{\text{all } i} SLQ^b(i) p_i^b | 2^{-b}}{\sum_{\text{all } b} 2^{-b}} \tag{3.5}$$

The weighted score gives more importance to lower cache line sizes pairs. This can be interpreted as higher relevance of these lower pairs because bigger cache line sizes bring massive data transfers. Usually, application with low spatial locality perform very bad on traditional systems with cache hierarchies because a small portion of data is utilized compared to the data loaded from the main memory to the caches.

3.2.3 Data level parallelism

Data-level parallelism (DLP) measures the average length of vector instructions that is used to optimize a program. DLP could be interesting for NMC when employing specific SIMD processing units in the logic layer of the 3D-stacked memory.
PISA can extract the amount of instruction-level parallelism for all the instructions (see *Figure 3.3*, CFG on the left) and additionally per instruction category such as control, memory, etc. (see *Figure 3.3*, CFG in the center). As shown in the CFG on the right in *Figure 3.3*, we extract the ILP score per opcode and call it as $ILP_{\text{specialized,opcode}}$ where opcode can be load, store, add, etc. This

metric represents the number of instructions with the same opcode that could run in parallel. Next, we compute the weighted average value for DLP using the weighted sum over all opcodes of $ILP_{\text{specialized,opcode}}$. The weights are the frequency of the opcodes calculated by dividing the number of instructions per code with the number of instructions.

$$DLP_{avg} = \sum_{\text{opcode}} ILP_{\text{specialized,opcode}} \frac{\#\text{instructions}_{\text{opcode}}}{\#\text{instructions}} \tag{3.6}$$

As the register allocation step is not performed at the level of intermediate representation, it is not possible to take into account the register consecutiveness in this score. However, we want to show the optimization opportunities for compilers distinguishing between consecutiveness of load/store instruction addresses. We represent this with two scores: DLP_1 without address consecutiveness; DLP_2 with addresses consecutiveness into account. To compute them we use the previous formula changing the $ILP_{\text{specialized,opcode}}$ value for loads and stores.

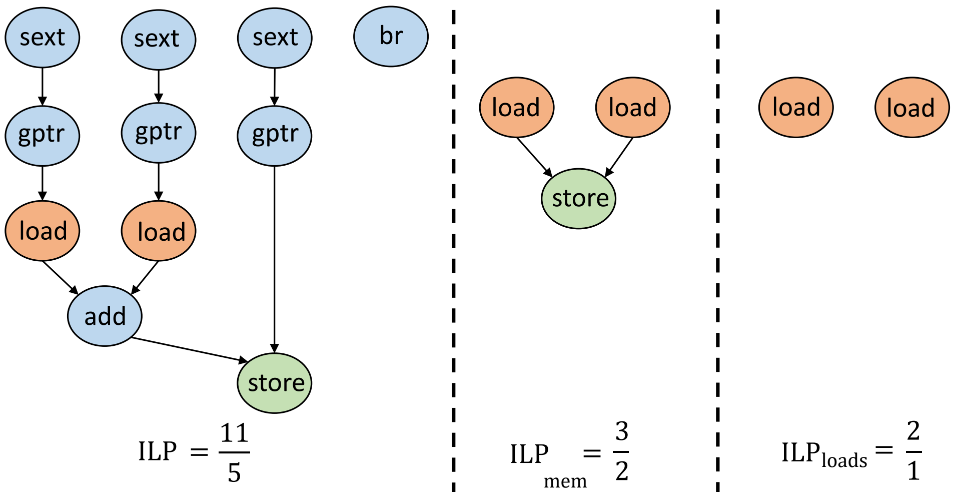


Figure 3.3: *Usually to compute ILP the control flow graph (CFG) is used, left side. The CFG in the center is used to compute the ILP per type of instruction. On the right our per opcode specialized CFG, which we label with $ILP_{specialized,opcode}$. The latter is used to compute the DLP_{avg} with Equation 3.6.*

3.2.4 Basic-block level parallelism

A basic-block is the smallest component in the LLVM's IR that can be considered as a potential parallelizable task. Basic-block level parallelism (BBLP) is a

potential metric for NMC because it can estimate the maximum amount of task level parallelism in the application. The parallel tasks can be offloaded to multiple compute units located on the logic layer of a 3D-stacked memory.
To estimate BBLP in a workload, we develop a metric similar to ILP and DLP. It is based on the assumption that a basic-block, which is a set of instructions, can only be executed sequentially. Since loop index count could put an artificially tight constraint on the parallelism, we assume two different basic-block scheduling approaches (see *Figure 3.4*): 1) all the dependencies between basic-block are considered; 2) we consider a smart scheduling, assuming a compiler that can optimize loop index update dependencies. The difference between the two approaches can give an idea, as in the DLP case, of the optimization opportunities for compilers. We compute the two scores derived from the two scheduling options using the following formula:

$$BBLP_{avg} = \frac{\#\text{instructions}}{\text{MaxIssueCycle}_{BBLP}}, \tag{3.7}$$

where $MaxIssueCycle_{BBLP}$ represents the cycle of the last executed instruction using the proposed scheduling approaches (red numbers in *Figure 3.4.b,.c*). $\#instructions$ represent the total number of instructions (see *Figure 3.4.a*).

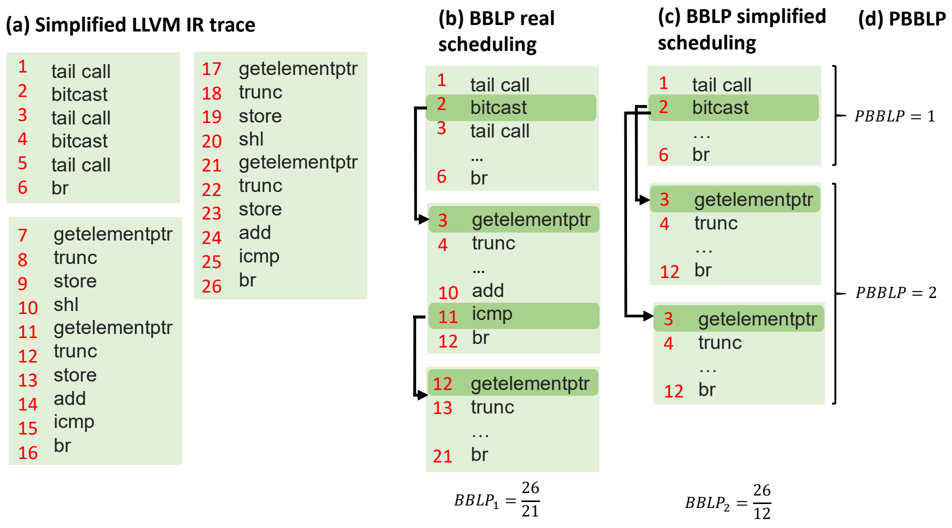


Figure 3.4: *BBLP/PBBLP methodology: a) example of a LLVM dynamic trace; b) real scheduling for BBLP computation taking in account all dependencies; c) simplified scheduling for BBLP computation not taking in account dependencies such as loop index update (in a) dependency between instruction 15 and 17); d) PBBLP values for each basic block (second and third block are a repeated basic block, since there is only a loop index dependecy, the PBBLP is equal to 2).*

We also aim to estimate the presence of data parallel loops. Data parallel loops consist of basic-blocks that are iterated without any dependencies among their instances. A fast and straightforward estimation can be done by assigning a value to each basic-block between 1 and the number of instances. When a basic-block has only one instance or all its instances have dependencies among them the score is 1. Instead, when all its instances don't have dependencies among them the value is maximal and equal to the number of instances. In practice, the score is within the range described above. Other assumptions we made are: skip index update dependencies and omit basic-blocks that are used only for index update.
After assigning a score to each basic-block ($PBBLP_{BB}$), we compute the weighted average value for PBBLP using the weighted sum over all scores ($PBBLP_{BB}$). The weights are the frequency of the basic-block instances calculated by dividing the number of instances per basic-block with the number of total instances.

$$PBBLP_{avg} = \sum_{BB} PBBLP_{BB} \frac{\#\text{instances}_{BB}}{\#\text{instances}_{\text{total}}} \tag{3.8}$$

Since this metric is an estimation we call it as potential basic-block level parallelism (PBBLP).

3.3 Results

We start by describing the system employed to evaluate the analysis result in in *Section 3.3.1*, and the target applications, see *Section 3.3.2*. The tool described in the previous *Section 3.1* is used to extract intrinsic application metrics in *Section 3.3.3*. Then, our next goal is to correlate the gathered metrics with performance on a real system. Therefore, we show in *Section 3.9* the correlation between the extracted application metrics and the performance on a NMC system.

3.3.1 System in use

To validate the relevance of our NMC specific metrics, we characterize the representative benchmarks using PISA-NMC and extract the numbers for memory entropy, spatial locality, data level parallelism, and basic-block level parallelism. We apply principal component analysis (PCA) [118], on the collected metrics to make it more understandable by reducing its dimensionality [91]. Next, we run the same benchmarks on an NMC system using a simulator and measure the improvement in energy-delay product and correlate this data with the output of PCA.
Figure 3.5 depicts the reference computing platform that we consider in this work. We run the applications both on a traditional Von-Neumann Architecture using the latest *IBM Power 9* [119] and on an NMC system based on hybrid memory cube (HMC). HMC memory is divided into several vertical DRAM partitions, called vaults, each with its own DRAM controller in the logic layer. In this work,

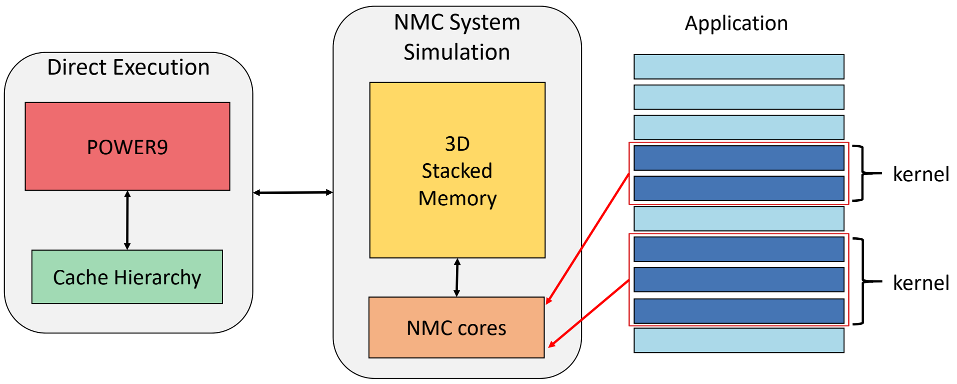


Figure 3.5: *High-level overview of the evaluation system. We directly execute the application kernels (dark blue boxes) on the IBM Power 9 and we evaluate the NMC performance on a NMC System Simulator (Ramulator).*

we model NMC PEs as in-order, single-issue cores with a private cache as proposed in previous work [120,121]. *Table 3.2* lists the details of the host and NMC system used in our experiments. We extract the power consumption with AMESTER[1] tool on Power 9. We simulate the NMC system on an extended version of the memory simulator *Ramulator* [77] including the processing units. Each processing unit is assigned to a vault and operates on the data assigned to that vault. We collect dynamic execution traces of the instrumented code with a Pin tool. We feed the acquired traces to Ramulator.

Table 3.2: *Host and NMC System Characteristics.*

Architecture	**CPU Used**	**Cache per core**	**Memory**
IBM Power9 (Host)	4 cores (SMT4) @ 2.3 GHz	L1 32 KB L2 256 KB L3 10 MB	DDR4, 32 GB RDIMM @ 2.7 GHz
NMC	32 single-issue in-order cores @ 1.25 GHz	L1-I/D 2-way 2 cache lines 64B per cache line	HMC, 4GB 8 stacked-layers, 32 vaults, 16-bit full duplex and SerDes I/O link @ 15 Gbps

[1]https://github.com/open-power/amester

3.3.2 Applications evaluation on NMC system

We compare our metrics to the performance achieved by running the applications on an NMC system. We perform a single-thread analysis to estimate our metrics and then evaluate the execution time on the considered architectures. Table 3.3 lists the parameter levels for the evaluated applications on the Power 9 and NMC system. We consider only single-thread analysis here to avoid the side effects of a multi-threaded analysis in metrics such as memory entropy and spatial locality, e.g., averaging the numbers from multiple threads tend to mask the true behavior of applications. Since the analysis trend is similar for different dataset sizes and the memory analysis is highly time-consuming we use smaller dataset than the one simulated for the NMC system in line with similar work on application characterization [117, 122]. *Figure 3.7* shows the energy-delay product (EDP) ratio between the IBM Power 9 and the NMC system we simulated. We use EDP as our major metric of reference in this analysis because both energy and performance are critical criteria for evaluating NMC suitability. Applications with EDP reduction less than 1 are not suitable for NMC.

Table 3.3: *Selected applications and parameters from the Polybench and Rodinia benchmark suites for the evaluation on the NMC system.*

Applications		**Parameters**	
Benchmarks	**Kernels**	**Param.**	**Values**
Polybench	atax, gemver, gesummv	dimensions	8000
	cholesky, gramschmidt, lu, mvt, syrk, trmm	dimensions	2000
Rodinia	bfs	nodes	1.0m
	bp	layer size	1.1m
	kmeans	data size	819k

3.3.3 Hardware agnostic characterization

We present the the characterization results of selected applications from **Poly-Bench** [123] and **Rodinia** [124] benchmarks (see *Figure 3.6*) employing the proposed metrics.

Memory entropy: in *Figure 3.6.a*, is strictly related to the dimension of the address space accessed by a workload. Indeed, applications with larger address space have higher entropy because they are accessing many different addresses. We also plot memory entropy changes at different granularity cutting the least-significant bits (LSBs) of the address to represent larger data access granularity. Furthermore, we highlight in Rodinia's applications the cut of 2 LSBs because they are accessing integer (4Byte locations). We notice that applications like `bp` and `gramschmidt` have higher values of entropy and they should benefit from NMC architectures. Contrariwise, the other applications have similar values except for `cholesky`, `bfs` and `kmeans`.

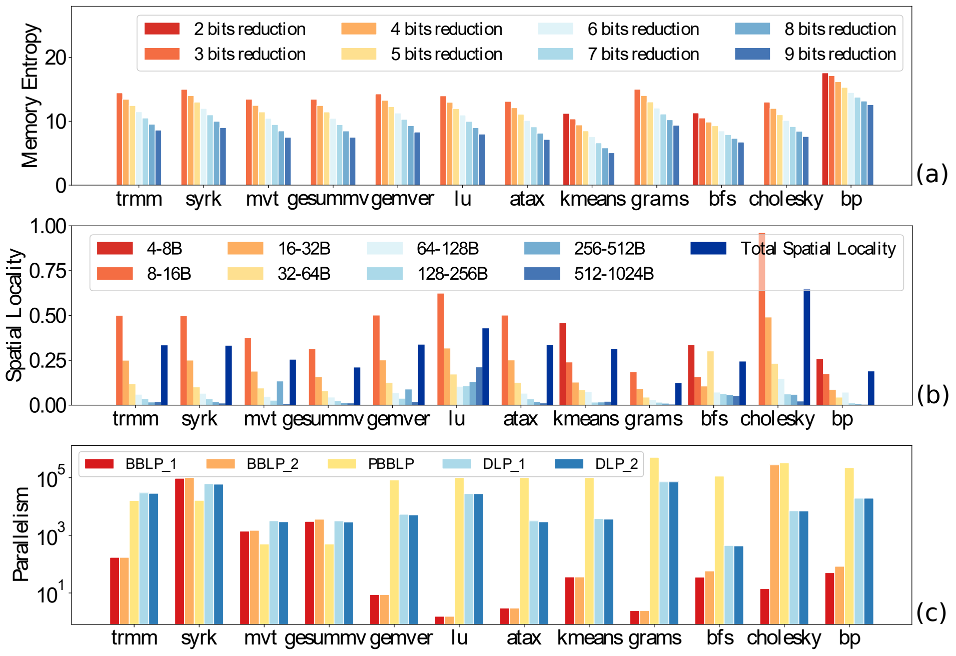


Figure 3.6: *Application characterization results:(a) Memory Entropy; (b) Spatial Locality; (c) Parallelism.*

Spatial Locality: we compute this metric related to memory behavior, we show it in *Figure 3.6.b*. As expected, we can distinguish different behaviors among the benchmarks. `bp` and `gramschmidth` show an interesting behavior with high entropy and low spatial locality. For instance, in `gramschmidt` accesses to the matrix are done by column and diagonally. However, the matrix allocation is done in a row-major order. These applications should be good candidates for NMC because they use a large address space with low locality. An opposite trend is detected for `cholesky`, where the entropy is one of the lowest value and the spatial locality is the highest value.

A considerable amount of applications show a spatial locality lower than 0.25 and they may benefit from NMC systems. However, applications with high spatial locality like `cholesky` could potentially also benefit from NMC mostly when increasing the data-set and consequently moving more data off-chip and exploiting SIMD architectures.

Parallelism: *Figure 3.6.c* shows the different parallelism characterization of

workloads. As expected in the Berkeley dwarfs for the data-level parallelism analysis [125], matrix multiplication based algorithms show the highest values. Moreover, the difference between the two proposed DLP scores seems to be very limited. Only small variations can be noticed, for instance in `trmm` and `syrk`. Here, the difference is due to loads/stores with non-sequential accesses and could be improved by a compiler exploiting data mapping techniques. Instead the BBLP scores show a significant difference for `cholesky` and limited differences for `bfs` and `syrk`. These results highlight possible parallelism optimizations that can be performed by compilers.
Finally, the $PBBLP$ score tries to highlight the presence of data parallel loops and gives an estimation of how much parallelism can be achieved using vectorization or loop unrolling strategies. Applications with high level of parallelism could benefit from NMC systems that provided multicores or SIMD architectures in the logic layer on top of the 3D-stacked memory.

3.3.4 Correlation between NMC Metrics and NMC Performance

Spatial locality in *Figure 3.6.b* provides insights on which application could be better for the NMC system we considered. Applications that show the lowest spatial locality such as `gramschmidt`, `bp`, `bfs` show a considerable EDP improvement (see *Figure 3.7*) using the NMC system. Contrariwise also `cholesky`, that has the highest spatial locality among the chosen applications, benefits from the NMC architecture.

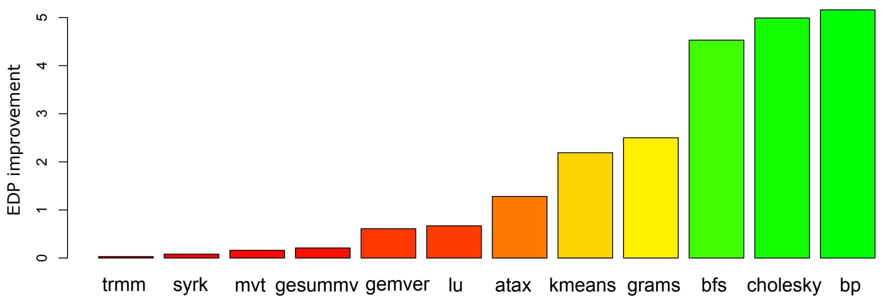


Figure 3.7: *Energy-delay product improvement on the NMC system in a red-to-green scale: red applications do not benefit from NMC acceleration, while the green ones perform better with this technology.*

Memory entropy in *Figure 3.6.a* gives similar insights. For instance, applications with the highest entropy such as `gramschmidth` and `bp` shows benefit executing on an NMC system. However, also applications with low entropy seem to benefit from NMC. Parallelism analysis in *Figure 3.6.c* highlights that most of the applications that benefit from NMC have the lowest values for $BBLP_1$ and a good level of DLP. However, there are some exceptions such as `lu`, that has the lowest BBLP values, and `bfs` that has the lowest DLP values. The above shows that a single metric can not explain NMC appropriateness. To get more insights into what combinations of metrics can predict NMC applicability, we apply PCA to our metric results.

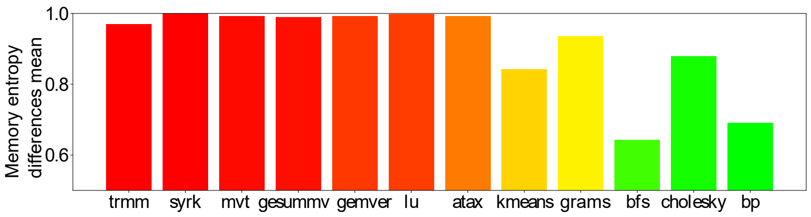


Figure 3.8: *The mean of the memory entropy differences: this additional metric shows high values for those applications that do not benefit from NMC acceleration.*

For this, we derive another metric from the memory entropy exploiting the granularity. For each application, we first compute the difference between each pair of consecutive memory entropy values with different granularities (see *Figure 3.6.a*, larger granularity represents larger cache line size). Then, we compute the average of these values that represents a spatial locality variation when increasing the cache line size. *Figure 3.8* shows this metric. This metric compared to the EDP values shows that the major part of the applications not suitable for NMC has the highest values.

Figure 3.9 shows the Principal Component Analysis (PCA) applied to the most promising subset of presented metrics. The PCA is used to detect the importance (arrow lenght) and the influence (arrow direction) over the new axes, the so-called principal components. We use 4 input features for the PCA: $BBLP_1$, $PBBLP$, $entropy\_diff\_mem$ (the value proposed above) and $spat\_8B\_16B$ (spatial locality doubling the cache line size from 8B to 16B). We highlight that all the applications that benefit from Power9 are in the II quadrant (top-left) except for `lu` that is in the III quadrant. In its code diagonal matrix accesses are present and they should be critical for traditional CPUs. It could be an NMC application candidate employing a larger dataset size. The applications that benefits from NMC are in the other quadrants. In particular `bfs` and `bp` seem having similar characteristic and are located in the I quadrant. Similarly `gramschmidt` and `kmeans` located in the IV quadrant. These metrics show good potential in discriminating NMC applications.

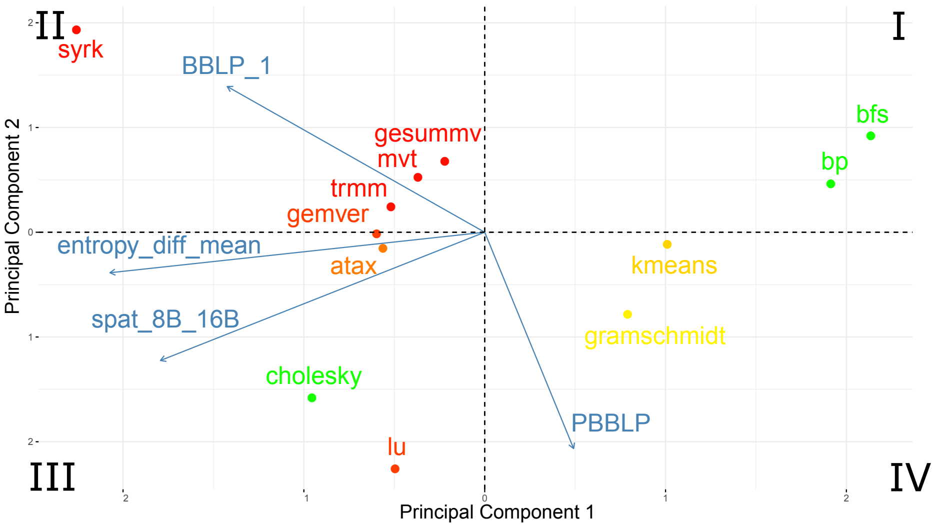


Figure 3.9: Principal Component Analysis using the added metrics. Blue arrows quantify the contribution and direction to the PCs. Most of the applications that do not benefit from NMC are constrained in the IV quadrant.

3.4 Related Work

Considerable effort has been already spent in realizing platform independent characterization tools. Cabezas [85] proposed a tool that can extract different features from workloads but has many limitations: the compiler community no longer supports the LLVM interpreter, and the target applications should be single threaded. Another tool has been developed by Shao et al. [22]. It can extract interesting metrics such as memory entropy and branch entropy. However, this tool has some limitations: it is based on the IDJIT IR (just-in-time compilation) that has compatibility problems with OpenMP and MPI, thus being limited to sequential applications. The state-of-the-art tool (called PISA) in workload characterization was presented by Anghel et al. [23]. PISA can analyze multi-threaded applications supporting the OpenMP and the MPI standards. PISA can extract the metrics such as instruction mix, branch entropy, data reuse distance, etc. We extended PISA with metrics directed towards NMC such as memory entropy and spatial locality, data-level and basic-block-level parallelism.
Existing studies primarily rely on hardware performance counters available in the modern processors to understand the memory access behavior of the applications and identify the kernels suitable for offload to NMC architectures [53,71,126–128].

Others have used a dynamic binary instrumentation framework like Pin [120] or estimation at the compile time [129,130] for the same purpose. PISA-NMC showed a different approach to workload characterization applied to NMC. We used a target-agnostic workload characterization technique to extract metrics directed towards NMC. Then, we used PCA and NMC simulation to show the relevance of the metrics proposed.

NAPEL: Assessing performance on NMC systems is complex due to the lack of available systems. Therefore, simulators are employed to determine the performance and energy consumption of NMC designs. However, simulation is often slow. In [21], we embed PISA in NAPEL, a high-level performance, and energy framework for NMC architectures. NAPEL (see *Figure 3.10*) mainly consists of ❸ a random forest model trained with ❷ microarchitectural features (extracted from an NMC simulator) and ❶ hardware-agnostic application features (PISA). Performance of unseen applications are predicted by using Ⓑ the trained model fed with Ⓐ hardware-independent analysis metrics.

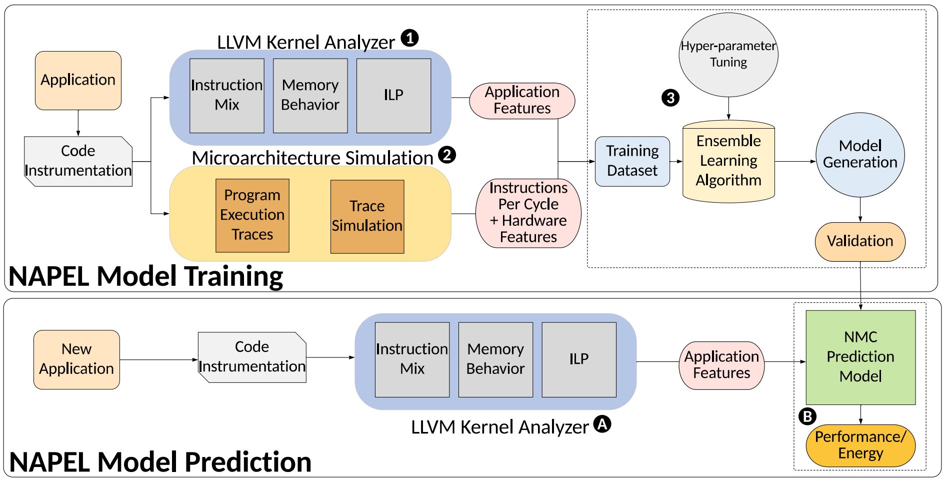


Figure 3.10: *NAPEL consists of two main phases: model training and model prediction. In the first phase (top), the application code is instrumented. Then hardware-agnostic features (1) and NMC simulation characteristics (2) are collected. A machine learning model is finally trained with the extracted data. An unseen application (bottom) is instrumented and analyzed with PISA in the model prediction. The model employs the independent hardware analysis to predict application performance [21].*

NAPEL can provide early design space exploration (DSE) much faster than a state-of-the-art NMC simulator. It can also accurately predict performance and energy estimation with an average error of 8.5% and 11.6%.

3.5 Summary and conclusions

Emerging computing architectures in their first stages of development, such as near-memory computing (NMC) lack proper tools for specialized workload profiling. In this scope, we present and extended version of PISA, a state-of-the-art application characterization tool, with NMC related metrics. Particularly, we focus on analyzing the memory accesses and parallelism behaviors: data-level parallelism, basic-block level parallelism, memory entropy, and spatial locality. By correlating the principal components of metrics as mentioned above with the energy-delay product of benchmark kernels on an NMC system, we show that PISA-NMC can help identify the kernels that can benefit from NMC in a platform-agnostic manner.

Limitations: Platform independent software analysis can be an appealing method to extract application features and accurately predict performance for emerging computing systems, e.g., near-memory computing [21]. However, this analysis methodology is typically time-consuming: based on the dataset size, 2 to 3 orders of magnitude slower than the plain application execution time. This heavily limits the design space exploration. Indeed, as shown in *Figure 3.10* in the prediction model, NAPEL employs PISA, which, as already mentioned, can be enormously time-consuming, although very precise in predicting energy delay product. A solution to improve this time consuming procedure is to employ hardware-dependent analysis in the ensemble machine learning model. In *Chapter 4*, we present NMPO, a high-level framework capable of predicting NMC offloading suitability by relying on hardware-dependent feature demonstrating how this approach is faster than the one shown in NAPEL. Indeed, NMPO has a similar structure to NAPEL, but it uses hardware-dependent analysis in the prediction loop. The latter analysis has a limited overhead compared to PISA, making the prediction step much faster. In other words, removing PISA from the prediction loop drastically improve the prediction model performance.

4 Ensemble machine learning for NMC offloading

Real-world applications have high performance and energy efficiency requirements. It is crucial to understand how these can be optimized by e.g., exploiting accelerators and new computational paradigms. Often workloads are bottlenecked by the data movement between the compute units and the main memory. Near-memory computing (NMC), a modern data-centric computational paradigm, can alleviate these bottlenecks, thereby improving the performance of applications. The lack of NMC system availability makes simulators the primary evaluation tool for performance estimation. This chapter proposes Near-Memory computing Profiling and Offloading (NMPO), a high-level framework capable of predicting NMC offloading suitability employing an ensemble machine learning model. NMPO predicts NMC suitability with an accuracy of 85.6% and, compared to prior works, can reduce the prediction time by using hardware-dependent applications features by up to 3 order of magnitude. This can be achieved by removing from the prediction loop the expensive hardware-independent application analysis.
The chapter starts by introducing additional notions on application characterization and NMC simulation in *Section 4.1*. Then, the NMPO framework and the hardware experimental setup are presented in *Section 4.2*. Therefore, we show the results and we evaluate them in *Section 4.3*. Finally, we present in *Section 4.4* the related work and in *Section 4.5* the conclusions.

This chapter is based on: S. Corda et al., "NMPO: Near-Memory computing Profiling and Offloading", DSD 2021

4.1 Background

This section reports the necessary background about performance monitoring counters (*Section 4.1.1*), NMC simulation (*Section 4.1.2*) and ensemble machine learning models (*Section 4.1.3*).

4.1.1 Application characterization

Key application features that are used later for taking offloading decisions can be collected in different ways. The quicker and easier way of evaluating an application on a traditional CPU is using hardware performance monitoring units (PMUs). Modern CPUs have specific programmable components programmed to gather information from different locations of the chip. Currently, a wide range of tools and libraries can be employed for this task, such as PAPI [74], LIKWID [131], and perf [73]. Perf is a ready-to-use utility available in most current Linux distributions. This utility collects an enormous amount of information from the analyzed application, such as cache misses, Clock cycles per Instructions (CPI), and floating-point operations. We summarize the main features that are collected in *Table 4.1* and used in the machine learning model.

Table 4.1: *Performance counters extracted with the perf tool on the Intel i9 9900K.*

Event name	Units		Event name	Units
power/energy-pkg/	Joules		L1-dcache-loads	countof
power/energy-psys/	Joules		L1-dcache-stores	countof
power/energy-ram/	Joules		L1-icache-load-misses	countof
uncore_imc/data_reads/	MiB		LLC-load-misses	countof
uncore_imc/data_writes/	MiB		context switch	countof
fp_arith_inst_retired	Gflops		App execution time	seconds
branch-instruction/branches	countof		LLC-loads	countof
branch-misses	countof		LLC-store-misses	countof
cache-misses	countof		LLC-stores	countof
cpu-cycles OR cycles	countof		branch-load-misses	countof
instructions	countof		branch-loads	countof
L1-dcache-load-misses	countof		Instructions/cycle	IPC

4.1.2 NMC simulation

Since NMC systems adoption is still not widespread, simulators are necessarily employed to determine their performance. Extended versions of Ramulator [21, 57, 132, 133] are utilized because of its easy extendibility, speed and accuracy. Ramulator is a cycle-accurate and portable memory simulator that simulates a wide range of modern DRAM technologies such as HBM (High Bandwidth Memory), HMC (Hyper Memory Cube), and WideIO. *Figure 4.1* shows a high-level representation of Ramulator.

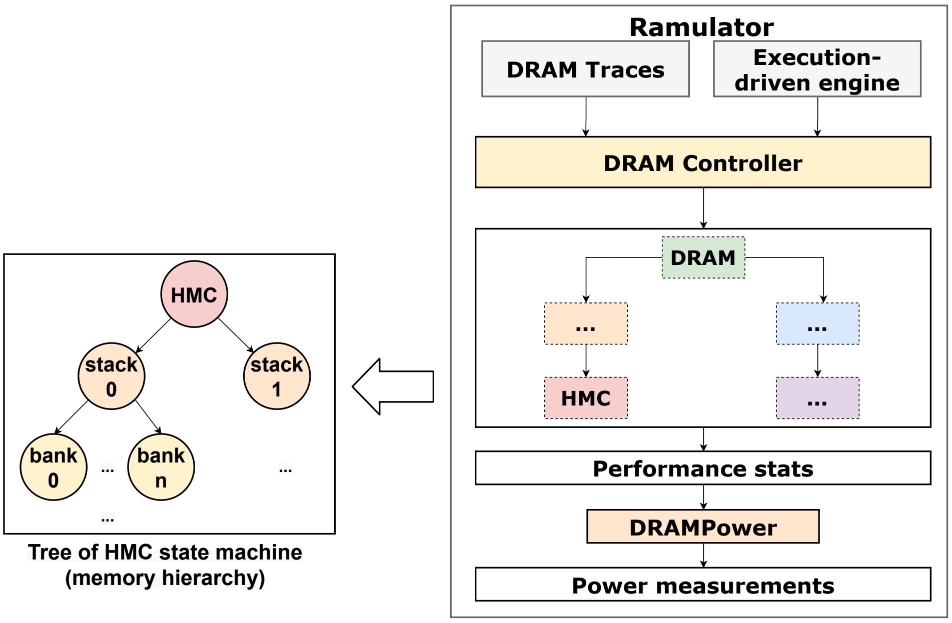


Figure 4.1: *High-level overview of Ramulator. It consists of a memory controller that takes as input the application simulation traces that can be in the format of DRAM traces or execution-drive engine (e.g. Gem5) generated traces. The DRAM is represented as a state machine. Ramulator outputs performance stats that can be integrated with DRAMPower to obtain power measurements.*

It consists of a memory controller that takes the simulation's input. This input can be a set of memory traces generated by a CPU simulator such as Zsim [80], which is called standalone mode, or it can be generated by an execution-driven engine such as Gem5 [79], which is named integrated mode. Ramulator's core consists of a tree of DRAM state-machines (left side of *Figure 4.1*), where each node is a class instance, such as HMC, that derives its properties from its parents' nodes. Each DRAM class has a hierarchy of banks, channels, ranks, etc., representing different nodes having a specific label as property. Ramulator-PIM (Processing-In-Memory), an extended version of Ramulator, can simulate computing units such as Out of Order (Ooo) cores on the logic layer of 3D-stacked memory.

For the evaluation of power consumption metrics Ramulator is integrated with DRAM power models such as DRAMPower [134]. In *Table 4.2* we summarize the main performance metrics that can be extracted using Ramulator-PIM.

Table 4.2: *NMC performance metrics available in Ramulator.*

Statistic	**Units**	**Category**
ramulator.cpu_cycles	cycle	Ramulator-PIM
ramulator.ipc	Instructions/cycle	Ramulator-PIM
ramulator.cpu_instructions	countof	Ramulator-PIM
ramulator.total_time	ns	Ramulator-PIM
Average Power	mW	DRAM Power
Total Trace Energy	pJ	DRAM Power

4.1.3 Ensemble machine learning

Complex decisions such as application offloading to suitable accelerators may require sophisticated tools such as machine learning (ML) prediction models. These models are usually trained on a section of the available features dataset and tested on the remaining part. Then, they are employed to predict, make decisions or classify an unknown dataset. While simple models such as a Decision Tree can be effective, in the case of many features, ensemble ML models are more accurate [135]. Ensemble ML models consist of several simple models trained on a different and random subset of the training dataset. The final decision, classification, or prediction is then made by evaluating all the simple models' results and selecting the most common outcome.

Random Forest (RF) [136] is an ensemble ML model that consists of a set of decision trees. RF uses either a categorical response variable, referred to in [137] as "classification", or a continuous response referred to as "regression". Similarly, the predictor variables can be either categorical or continuous. The decision trees are partitioned based on binary recursion. The predictor space uses a sequence of binary splits to partition. The root node contains the whole list of predictors. The splitting criterion gives a measure of "goodness of fit" (regression) or "purity" (classification) for a node, with large values representing poor fit (regression) or an impure node (classification).

RF model performance is boosted by tuning the hyper-parameters, which are characteristics of the model that can impact model accuracy and computational efficiency. These values are set before fitting the model and optimized through trial and error methods like grid search and random search. Multiple models are fitted with several hyper-parameter value sets, their performances are compared, and the best performing one is chosen. The popular hyper-parameters tuned for Random Forest models are; the number of decision trees (N_estimators), the number of features to resample (Max_features), the depth of each tree in the forest (Max_depth), the minimum number of samples required to split each node (Min_samples_split) and the minimum number of samples required for each leaf (Min_samples_leaf).

We are going to use ensemble machine learning to predict near-memory computing offloading suitability of unseen applications.

4.2 Framework

NMPO, a high-level framework based on ensemble machine learning developed to predict near-memory computing offloading suitability, is described in *Section 4.2.1*. *Section 4.2.2* presents the experimental setup employed for this work, in particular showing details on the architecture and benchmarks used.

4.2.1 NMPO

NMPO (see *Figure 4.2*) consists mainly of two separate parts: the first one for characterizing the application and training the machine learning model and the second one where the offloading decision is taken by employing the ML model's prediction result. More precisely, in the first phase, the applications are characterized on the host system (❶) employing PMUs and collecting information as reported in *Table 4.1*. Then, the applications are simulated on the NMC system (❷), using Ramulator and DRAMPower to obtain the performance measurements (see *Table 4.2*). The performance metrics gathered from these steps are applied to evaluate the NMC offloading suitability. Thus, we label the data for the machine learning model by our criteria of offloading based on Energy-Delay-Product speedup, which is computed as follows:

$$\text{EDP\_improvement} = \text{Host\_EDP}/\ \text{NMC\_EDP} \tag{4.1}$$

Accordingly, for the collected training data, we label the offloading decision as "yes" if EDP_improvement > 2, "maybe" if $1 <$ EDP_speedup > 2 and "no" if EDP_improvement < 1. Finally, the machine learning model is trained (❸) using the previous analysis metrics. We employ k-fold validation to evaluate the ML model. For each of the K folds, the model is trained on the remaining (K - 1) folds, which are considered training data and tested on the remaining data or the left-out fold, which serves as the testing data. The performance of the machine learning model is evaluated as the average performance over K-iterations of cross-validation.
The hyper-parameters, which are the ML algorithm variables, are tuned to optimize the prediction model's accuracy. The application offloading of the unseen application is performed by first profiling the application 🅐, similarly to ❶, on the host system with PMUs (see *Table 4.1*). Then, the trained ML model uses the extracted features to predict the offloading decision on an NMC system 🅑. More precisely, since the key feature is the Ramulator IPC (see *Figure 4.7*), the ML model predicts this key feature for unseen application by employing an RF regression model by using only the host system characterization and later predicts the NMC suitability by classifying the results.

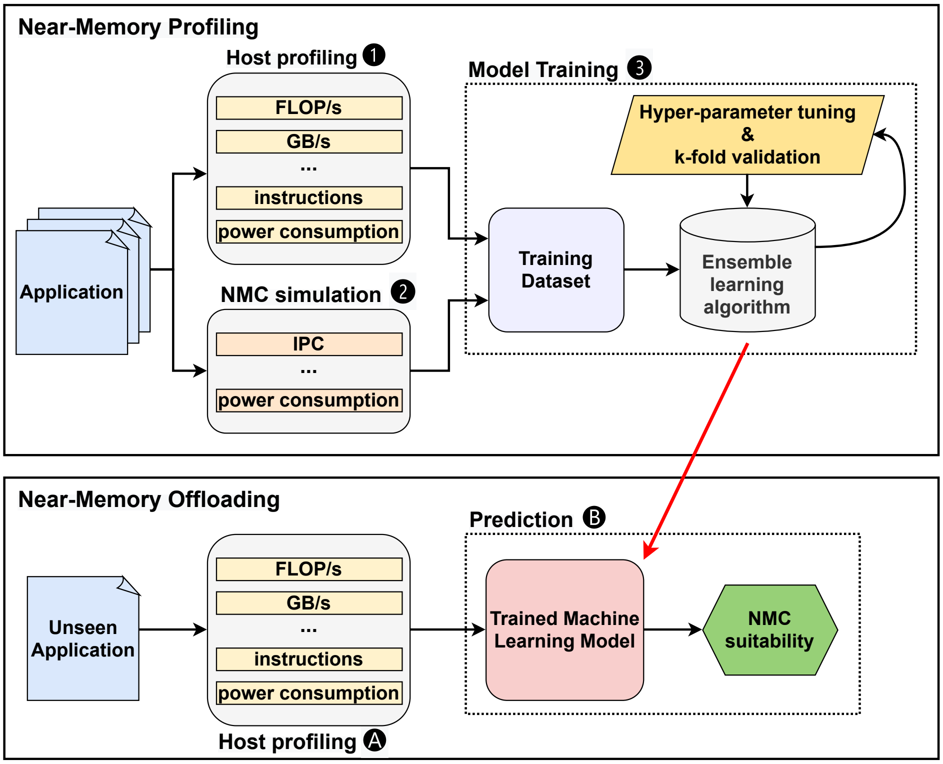


Figure 4.2: *Near-Memory Computing Profiling and Offloading (NMPO) overview. In the first phase (top) the framework performs an application profiling on the host by employing perf, a hardware-dependent tool, and an NMC simulation with Ramulator. Then (bottom), an ensemble machine learning model is trained with the previously collected data. In the second phase unseen applications are characterized used the same hardware-dependent tool to extract metrics that are feed to the machine learning model, which finally predict the applications' NMC suitability. Note: this is very similar to NAPEL [21]. The key improvement is removing the Platform-Independent Software Analysis from the prediction loop, which can takes up to hours or days based on the application and dataset employed.*

The performance of the machine learning model evaluates as the average performance over K-iterations of cross-validation. For example, let the RF ensemble model compute the regression error in predicting the kth part using RMSE and cross-validation score (CV) as:

$$RMSE_k = \sqrt{\frac{\sum_{i \in kth \text{ part}} (\text{Predicted}_i - \text{Actual}_i)^2}{N}} \tag{4.2}$$

$$CV = \frac{1}{K}\sum_{k=1}^{K} RMSE_k \tag{4.3}$$

We shaped the NMC offloading decision as a classification problem, where the key error metric is accuracy that corresponds to the ratio of correct prediction and the total number of predictions:

$$\text{Accuracy} = \frac{\text{Number of correct predictions}}{\text{Total number of predictions}} \tag{4.4}$$

We also applied the confusion matrix as an alternative tool to better visualize the same information. In the confusion matrix, each row of the matrix represents the instances in a predicted class, each column represents the instances in an actual class. The confusion matrix is named since it makes it easy to see if the system confuses one class for another.

4.2.2 Experimental Setup

Our experimental setups consist of a host processor (see *Figure 4.3*), an Intel i9 9900K, and an NMC system with out-of-order (OoO) cores placed on the logic layer of the HMC memory. The latter is simulated by Ramulator. The details of the host and NMC system are presented in *Table 4.3*.
The applications are profiled on the host system five times, extracting mean values with the perf package available with Ubuntu 18.04. The NMC system is simulated with Ramulator-PIM [56] once, since the results do not vary in different runs. Power and time parameters for HMC are derived from [134, 138] and fed to the NMC simulator. As benchmark, we selected a set of application from Polybench since it consists of simple mathematical operations extensively used in modern applications, and are also commonly used in NMC related work [129]. We selected implementation of the benchmark using OpenMP [123, 139], in order to exploit parallelism on the CPU.

Table 4.3: *Architectural parameters of the Host and NMC system.*

Host system	
Intel i9 9900K	8 cores, 2 threads per core, 1 socket, 4.7 GHz, 16 MB L3 cache, 64 GB DDR4 2666 MHz,

NMC system	
Ramulator	8 single issue OoO cores 1.25 GHz 2-way, 2 cache-lines, 64 B per cache-line 32 vaults, 8 stacked-layers, 256 B row buffer, 4 GB HMC

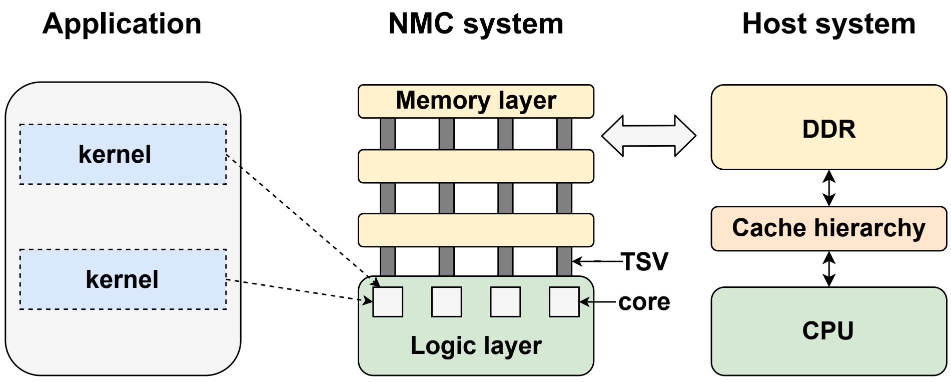


Figure 4.3: *System overview: application's kernels are evaluated on a host system, which consists of a traditional CPU and a simulated NMC system. The latter consists of a 3D-stacked memory (HMC) with small cores on the logic layers, which communicates with through-silicon vias*

Aside from the synthetic Polybench benchmarks, we use the current state-of-the-art gridding, and degridding algorithm for radio-astronomical imaging Image Domain Gridding (IDG) release 0.7 [4,140]. As shown in *Figure 4.3*, we analyzed only the kernel of interest. While Polybench applications have just one kernel, IDG contains different kernels such as gridder and degridder. The benchmarks, their parameters, and the value associated with the different dataset sizes are listed in *Tables 4.4* and *4.5*. The datasets are carefully chosen to be large enough to generate DRAM accesses and evaluate whether the application is really suitable for NMC. We also reported the time spent by the machine learning (ML) for the training, hyper-tuning and prediction, and the Ramulator simulation time for collecting training data (RT).

4.3 Experimental Results and evaluation

We discuss the results of application profiling and offloading. Further, empirical evidence in terms of validation and error metrics of the prediction models is presented. Finally, the prediction models are applied to the test cases for identifying the NMC suitability for a target application, thus aiding the users in early design stage explorations.

4.3.1 Application profiling

This stage provides the training data required to build and test our machine learning model described in *Section 4.2.1*.

Table 4.4: *Selected applications and their description from Polybench. The last two applications are real-world radio-astronomical imaging kernels.*

Application	
Name	Task
atax	Computes AT̂ times Ax
chol	Decomposes a matrix to triangular matrices
doit	Multiresolution ADaptive NumErical Scientific
gemv	Multiple matrix-vector multiplication
gesu	Summed matrix-vector multiplications
mvt	Matrix Vector Product
syrk	Symmetric rank k update
syr2k	Symmetric rank 2k update
trmm	Triangular matrix multiplication
grid	Radio-astronomical visibilities gridder
degrid	Radio-astronomical visibilities degridder

Table 4.5: *Parameters of the applications presented in* Table 4.4 *and timing of the machine learning model (ML is the time needed to train the model and RT is Ramulator simulation time). The applications are run with 8 and 16 threads to increase the available dataset points for the machine learning model. Ramulator was not able to simulate real-world motifs (gridder and degridder) with meaningful dataset sizes. Therefore, we report N.A. (not available).*

Application	**Datasets sizes**								**Time (min)**	
	1	2	3	4	5	6	7	**Test**	ML	RT
atax	4000	6000	8000	10000	12000	14000	16000	17000	3.25	180
chol	1024	1500	2000	2200	2600	3000	3400	4000	6	720
doit	75	100	128	150	200	256	300	350	6.25	5760
gemv	4000	6000	8000	10000	12000	14000	16000	18000	7.15	186
gesu	4000	6000	8000	10000	12000	14000	16000	18000	8.35	202
mvt	4000	6000	8000	10000	12000	14000	16000	18000	7.56	173
syrk	1024	1500	2000	2500	2750	3000	3500	4000	9.32	4568
syr2k	1024	1500	2000	2500	2750	3000	3500	4000	8.4	4898
trmm	1024	1500	2000	2500	2750	3000	3500	4000	7.35	5280
grid	128	256	512	2048	2560	3072	3584	4096	8.15	N.A.
degrid	128	256	512	2048	2560	3072	3584	4096	8.32	N.A.

Applications with chosen datasets sizes in *Table 4.5*, are profiled to collect various statistics from perf, Ramulator-PIM and DRAMPower as discussed in *Section 4.1.1*. The roofline model [65, 141] is a method for capturing the compute-memory ratio of computation and determines if the application is compute-bound or memory bound. The roofline model shows the application's achieved performance (GFLOP/s) and arithmetic intensity (FLOP/Byte) against the machine's maximum achievable performance.

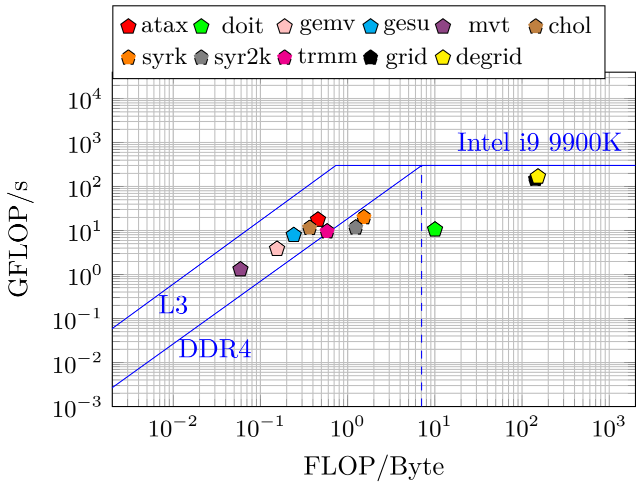


Figure 4.4: *Roofline plot of test cases using 16 threads. Most of the applications are bounded by the memory performance, in particular the last-level cache (L3). The considered radio-astronomical imaging kernels (gridder and degridder) and doit are compute bound. We use hardware performance counters to collect FLOPs and Bytes from DRAM. Especially DRAM counters are not precise as the one employed for FLOPs and may be influcende by different factors such as prefetching and system kernel operations.*

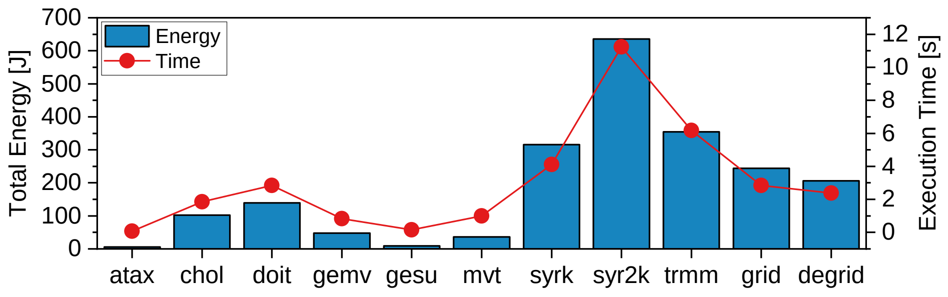


Figure 4.5: *Execution time and total energy of test cases on Intel i9 using 16 threads. We can also notice, as expectedm a very sharp correlation between execution time and enegy.*

In *Figure 4.4* the roofline model of 16 threads test datasets is plotted as an example. Application such as gridder, degridder and doitgen are compute-bound;

Symmetric rank update algorithms (syrk and syr2k) are in the DRAM-bound region, whereas the rest of the applications are L3-cache bounded. This tool helps to demonstrate the heterogeneity of the benchmark employed.
In *Figure 4.5* Total energy (J) vs Execution time (s) is depicted for all test cases for 16 threads showing the above-mentioned applications heterogeneity. The proposed work uses the energy-delay product (EDP) of host and NMC, where energy is the total energy consumption by the cores and delay is the amount of time for executing applications. Then, we compute the EDP improvement (*Figure 4.6* shows only the EDP improvement for the test cases using 16 threads) for each training dataset. Application with lower arithmetic intensity (highly memory bound) such as mvt benefit more from NMC offloading.

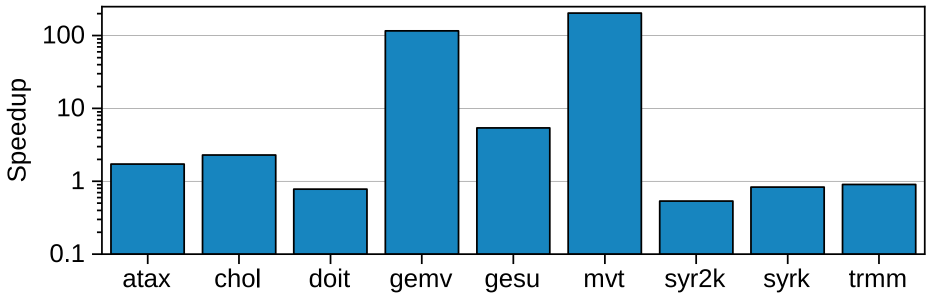


Figure 4.6: *EDP Speedup of Polybench test cases using 16 threads. Compared to Figure 4.4 only the L3-bounded applications benefit from near-memory computing. Gridder and Degridder values are not available because Ramulator was not capable to simulate these real-world application with meaningful datasets.*

4.3.2 Application offloading

We present further details on how we selected the features of our machine learning model. Then, we proceed by evaluating the accuracy of the model in predicting NMC offloading suitability. We conclude this section by showing how time-consuming PISA is compared to hardware-dependent tool such as perf.

Feature selection

Feature selection methods are intended to reduce the number of input variables to ones that are the most beneficial for a model to predict the target variable. This technique is employed to improve estimators' accuracy scores or boost their performance on very high-dimensional data sets. In our analysis, we selected the essential features using Pearson correlation [142]. It is represented by a number between -1 and 1 that indicates the extent to which two variables are linearly correlated. A value closer to 1 implies a stronger positive correlation, and a value closer to -1 indicates a negative correlation.

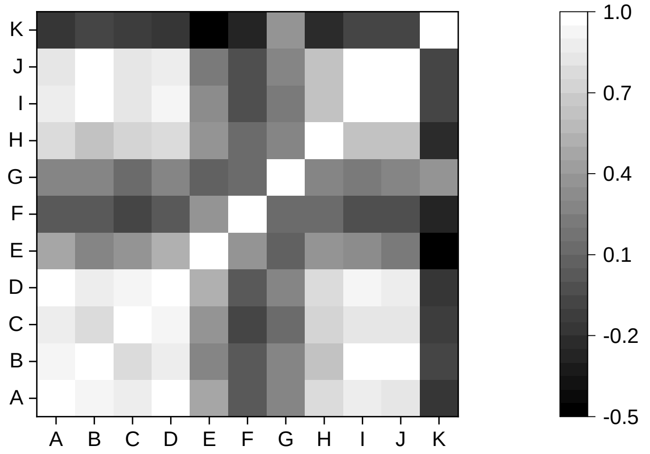


Figure 4.7: *Correlation plot of input features (see legend in* Table 4.6*). The Speedup (K) is stricly related to Ramulator IPC (G).*

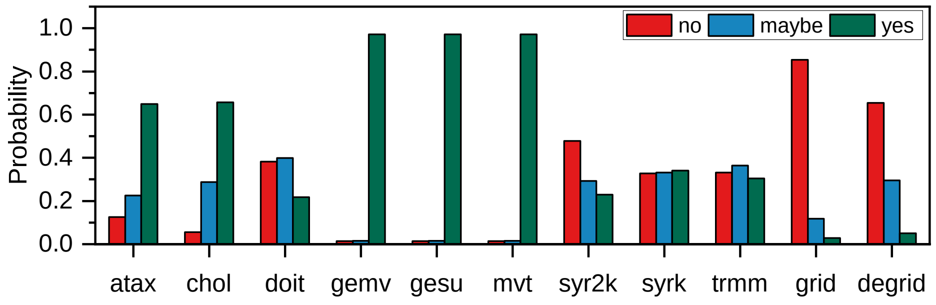
Table 4.6: *Legend for* Figure 4.7*.*

Feature	**Symbol**
Host Total energy (J)	A
Host EDP	B
Host Total DRAM access (GB)	C
Host FLOPs	D
Host GFLOP/s	E
Host FLOP/B	F
Ramulator IPC	G
Ramulator Total Time (ns)	H
Ramulator/DRAMPower Total trace energy (pJ)	I
Ramulator EDP	J
Speedup	K

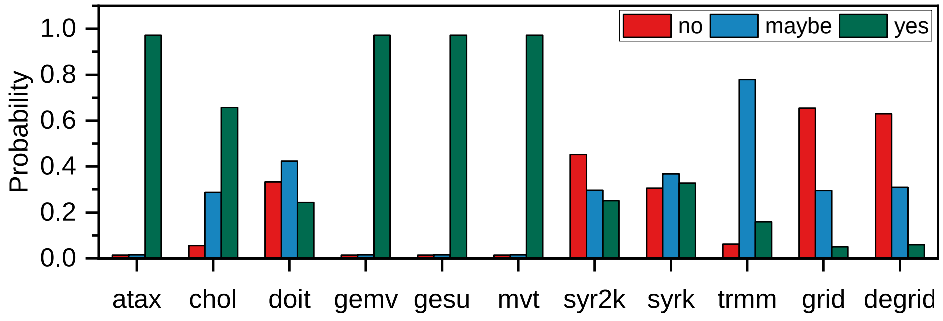
In *Figure 4.7*, we show the correlation of the main features we used in this work. It may be easily visible that the correlation is equal to 1 for the same metric, while in the other cases it is lower. Ramulator IPC is a key factor for making offloading decisions, and indeed it has the highest correlation with the EDP speedup. Since it is time-consuming to run Ramulator each time for a new unseen application or application with a different data set, we deploy an RF regression model to predict the Ramulator IPC and consequently predict the NMC suitability classification. This step is quicker than NMC simulation and enables early design exploration of unseen applications.

NMC suitability prediction

After the model is trained, validated and tuned, the final step is to test it on an unseen application. Similarly to [21], we trained the model using the data of all the application *excluding* the one the model will predict. In this manner, the prediction will be more realistic, and the application can be considered *unseen*. Since Ramulator is time-consuming and, in particular, it takes several days to simulate the radio-astronomical imaging algorithms, even with a small image such as 128x128 pixels, we used only Polybench applications for the training (excluding the predicted application if necessary). In particular, for this small dataset, more than 144 hours are necessary and large disk space is required, tipically a few terabytes.



(a)



(b)

Figure 4.8: *Model probability of predictions: (a) 8 threads, and (b) 16 threads. The offloading suitability are labelled using three categories: offload (yes), not offload (no), not sure (maybe).*

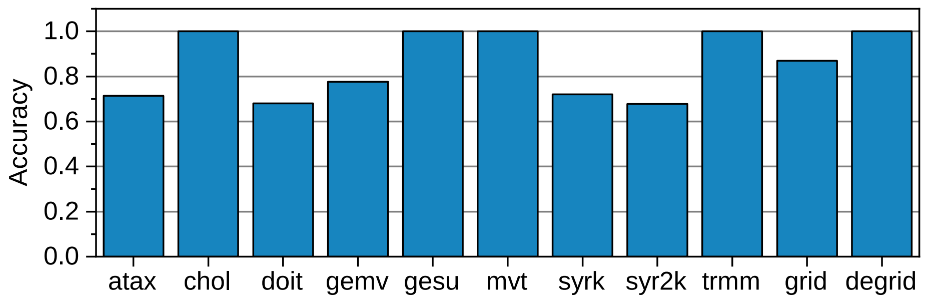


Figure 4.9: *Accuracy of offloading using NMPO, in average 85.6%.*

Furthermore, we simulate the above-mentioned small dataset for Gridder and Degridder, which are well-known compute-bound application [4] and will not benefit from NMC in any case to prove the unsuitability of these kernel for NMC offloading. Indeed, their EDP speedup is small (close to 0).
The machine learning model classification probability for the test cases is reported in *Figure 4.8* for both 8 and 16 threads test cases. For instance, we observe that applications with the lowest arithmetic intensity such as gemver, gesumm and mvt are the predicted with 100% accuracy. These are also the ones that benefit more from NMC offloading. The overall model accuracy is reported in *Figure 4.9* per applications. While some applications have a 100% accuracy, some of them are below 80%. In average, the accuracy is 85.6%.

Improved estimation for training time

Similar to [21] the main bottleneck in these design space explorations is usually located in the training phase, where the NMC system must be simulated. This procedure usually can take days for a single application for real-world datasets. Furthermore, in [21] the prediction phases consist in characterizing the application employing PISA [23]. However, PISA is much slower than PMUs and for specific applications needs more than 64 GB of DDR4, making this step really challenging. We report in *Figure 4.10* the execution time speedup of perf compared to PISA. We employed the datasets reported in *Table 4.7*, which are smaller compared to the ones in *Table 4.5*. Nonetheless, perf is already much faster than PISA for small datasets. This performance difference becomes even larger by increasing the datasets size. We can notice 2 to 3 order of magnitude improvement comparing perf to PISA, thus making the use of perf for the prediction phase more convenient.

4.4 Related Work

Near-memory computing past works focused mainly on selecting specific memory-bound applications and optimize them with custom architectures on the logic-layer

Table 4.7: *Application's datasets employed to compared PISA and perf. We also report the execution time of the two characterization tools. PISA is definitely time consuming.*

Application	Dataset	perf time [s]	PISA time [s]	Speedup
atax	2000	0.23	503.85	2190x
chol	512	0.16	28.01	175x
doit	64	0.27	20.78	77x
gemv	2000	0.26	55.62	214x
gesu	2000	0.32	69.27	216x
mvt	2000	0.24	356.16	1484x
syrk	512	0.54	201.25	373x
syr2k	512	0.86	416.44	484x
trmm	512	0.36	140.6	391x

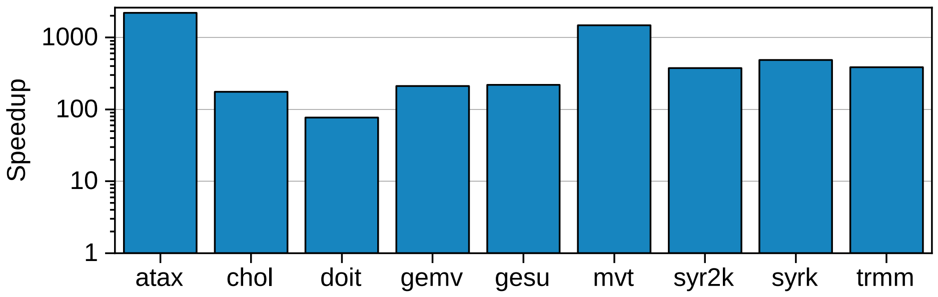


Figure 4.10: *Perf vs PISA execution time comparison. PISA is slower from 2 to 3 order of magnitude compared to perf.*

of 3d-stacked memory [14]. A few of them focused on offloading mechanisms or performance prediction to decide if the NMC system's scheduling is beneficial. We summarize the main related work on application offloading on NMC systems in *Table 4.8*.

Zhang et al. [71] employ a performance prediction model to decide how to schedule applications on their GPU-based NMC architecture. Ahn et al. [130] propose an offloading ad data mapping mechanism hidden to the programmer. This compiler-based mechanism can efficiently schedule workloads on their NMC-GPU system employing metrics such as memory bandwidth cost-benefit and memory mapping benefits. Hsieh et al. [143] propose an ISA extension to support NMC execution on an NMC system consisting of OoO cores and HMC. The programmer must use the proposed ISA extension to offload specific instruction to the NMC architecture. Hadidi et al. [144] extend GraphPIM [146] propose a compiler-

Table 4.8: *NMC offloading related work.*

Name	**Year**	**Offloading**	**Accelerator**	**Memory**
Zhang et al. [71]	2014	estimation model	GPU	HMC
Ahn et al [130]	2015	compiler and run-time	GPU	HMC
Hsieh et al. [143]	2016	run-time	Ooo cores	HMC
Hadidi et al. [144]	2017	compiler	Fixed function units	HMC
Ahmed et al. [145]	2019	compiler	Fixed function units	HMC
Corda et al. [31]	2019	PCA	in-order cores	HMC
Singh et al. [21]	2019	ML model	in-order cores	HMC

based mechanism for instruction offloading on CPU/GPU-NMC systems. Ahmed et al. [145] propose a compiler-based mechanism able to detect code sections that reduce the off-chip data movement when accelerated on a CPU connected to HMC. Ad detailed in *Chapter 3* employ PISA-NMC [30, 31], an extended version of PISA capable of extracting metrics related to memory and task parallelism, to evaluate the correlation of these metrics and the NMC offloading suitability using the Principal Component Analysis (PCA). Singh et al. [21] design a high-level framework for predicting unseen application performance on an NMC system. This framework consists of a tuned random-forest model trained with hardware-independent feature and performance on an NMC system with HMC and in-order cores. While the model is capable of predicting the energy-delay-product accurately, prediction is slow. Indeed, this prediction needs to gather the hardware-independent feature of the unseen application using PISA [23], which may take from 2 to 3 orders of magnitude compared to the application's execution time in the host system as we show in *Section 4.3.2*. We use the hardware-dependent application features collected with a small execution time overhead to predict the NMC offloading suitability to overcome this critical issue. Furthermore, while in [21] specific datasets are so small that they cannot be sampled by perf the PMUs (execution time lower than 0.001s), we selected large datasets that can generate DRAM traffic. This makes it possible to evaluate which applications are suitable for NMC offloading when accessing external DRAM.

Table 4.9: *Performance prediction related work.*

Name	**Year**	**ML model**	**Architecture**
Joseph et al. [147]	2006	Linear Regression	CPU
Calotoiu et al [148]	2013	Empirical model	CPU
Bailey et al. [149]	2014	Linear Regression	CPU/GPU
Wu et al. [150]	2015	ANN	GPU
Mariani et al. [151]	2017	Random Forest	Cloud HPC
Singh et al. [21]	2019	Random Forest	NMC

Performance prediction of unseen applications on specific architectures is a widely researched topic. However, just some of them focus on NMC. Indeed, as shown in *Table 4.9*, most of them focus on CPU and GPU as target offloading architecture. Concerning the machine learning model employed, in past work, linear regression, ANN and random forest have been employed with different tuning options. Similar to Singh et al. [21] and Mariani et al. [151] we employ the random forest algorithm because it can achieve higher prediction accuracy.

4.5 Summary and conclusions

We present NMPO, a high-level framework based on ensemble learning models and hardware-dependent profiling that facilitate quick and precise predictions to offload suitable applications to NMC kernels. This framework aids in the early design stage exploration of unseen applications on modern DRAMs like HMC. Unlike slow simulators, NMPO employs an ensemble learning technique called Random Forest with hyper tuning to speculate the offloading of an application. Furthermore, NMPO is much faster than the current state-of-the-art NMC simulator, and other machine learning-based frameworks with platform-independent profiling since hardware-dependent characterization used in NMPO has far less execution time overhead than hardware-independent ones. Thus, NMPO with 85.6% accuracy, quicker analysis and user-friendliness is the go-to ML-based framework for early design stage exploration.

Limitations:. Although NMPO can predict the NMC suitability of unseen applications quicker than the state-of-the-art, it is not able to accurately predict performance. Indeed, NMPO shapes the prediction into a classification problem, and it labels applications by their suitability or unsuitability for NMC offloading. Hardware-independent characterization can produce more accurate analysis if targeting an unavailable architecture because it analyzes the intrinsic behavior of the application. Moreover, it is not affected by the system overhead compared to hardware-dependent characterization. However, hardware-dependent techniques have minimal overhead and, in most cases, are precise for individuating optimization and system bottleneck on specific architecture, and possibly help in making offloading decisions. The next chapters show the importance of application characterization and different optimization techniques to make scientific computing, such as radio-astronomical imaging, highly performing and energy-efficient.

5 Large 2D FFT radio-astronomy

Scientific computing applications have high performance and energy efficiency requirements. In particular, radio-astronomical imaging needs to process TeraByte per second of radio-astronomical data and execute ExaFlop per second to construct a high-resolution map of the sky in realtime. Such large images make memory bottlenecks arise in modern radio telescopes like the Square Kilometre Array (SKA) processing. Indeed, we characterize the prediction and inversion steps (see *Section 2.3.3*, which consist of a gridding/degridding operations followed by a large 2D inverse/direct FFT. We identify that a sub-module performing a two-dimensiona fast Fourier transform (2D FFT) become a critical bottleneck by analyzing the Image-Domain Gridding, a state-of-the art gridder/degridder algorithm. We evaluate the application properties on an IBM Power 9 by applying the CPI (Cycles per instruction) and the roofline model. Then we present an NMC approach on FPGA for 2D FFT that can outperform a CPU by up to a factor of 120x and performs comparably to a high-end GPU while using less bandwidth and memory.

As shown in *Figure 5.1*, while some kernels, Gridder and Degridder, perform quite well, even when increasing the image size, we observe that the large 2D FFTs become the application bottleneck since it is memory bandwidth bound, and it does not reach peak performance. *Figure 5.1* shows the contribution of FFT in the execution time of IDG for different image sizes (also called grid sizes).

This chapter is based on: S. Corda et al., "Near memory acceleration on high resolution radio astronomy imaging", MECO 2020.

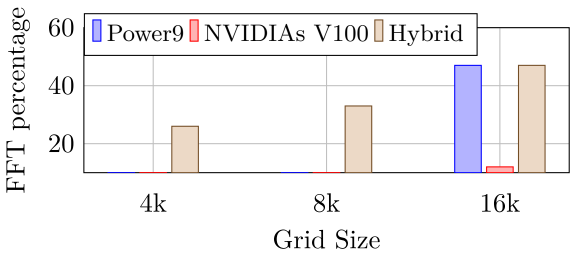


Figure 5.1: *Percentage of the IDG execution time spent on 2D FFT. The maximum (100%) is the total execution time of IDG. The Hybrid system consists of running the FFT on the CPU and the Gridder/Degridder on GPU. This solution has been adopted because for larger image size (e.g. 32k points per dimension) the GPU does not have sufficient memory.*

The chapter starts with a background information of the CPI breakdown analysis in *Section 5.1* and the methodology employed in *Section 5.2*. Then, we present the results in *Section 5.3* and the related work in *Section 5.4*. Finally, the chapter concludes with *Section 5.5*.

5.1 CPI Breakdown Analysis

CPU architectures, such as Intel, can be studied employing approaches/tools such as Top-Down [96] or Intel VTune [64]. Instead, IBM Power architecture does not have great application characterization tool support. Therefore, a fair amount of effort must be spent on providing these tools and methods to IBM Power CPUs. Indeed, in this work, we focus on the IBM Power9, which can be analyzed using the same methodology presented in [152] for IBM Power8.

PMUs are programmable components contained inside each microprocessor core on the chip. They are used to collect and filter information gathered from various aspects of the chip and they can attribute the events to the threads within the core. Power9 supports around 1000 PMU[1] events that can be monitored. The CPI Breakdown consists of creating a breakdown of the total run cycles in different categories, e.g. stalls in load/store units, to understand where the application is spending most of the time, thus being able to detect application bottlenecks. A simplified representation, containing the most interesting PMUs for memory bottlenecks (see *Section* 5.2), of the CPI breakdown is reported in *Figure 5.2* and a description of the used PMU is given in *Table 5.1*.

[1]https://wiki.raptorcs.com/w/images/6/6b/POWER9_PMU_UG_v12_28NOV2018_pub.pdf

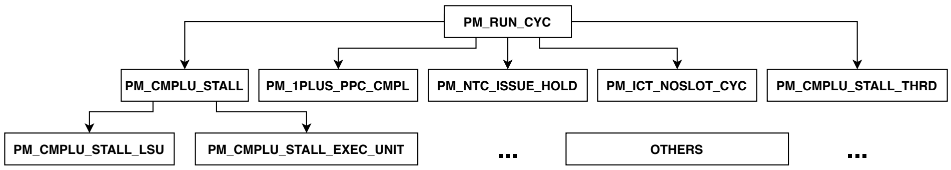


Figure 5.2: *Power9 CPI Breakdown tree [152]. The total cycles (PM_RUN_CYC) are breakdowned into different categories such as cycles while the CPU stalls (PM_CMPLU_STALL), which in turn can be split into cycles waiting for the load-store units (PM_CMPLU_STALL_LSU) or execution units (PM_CMPLU_STALL_EXEC_UNIT).*

Table 5.1: *IBM Power9's Performance Monitoring Units (PMUs) description.*

PMU	**Description**
PM_RUN_CYC	Run cycles
PM_CMPLU_STALL	Nothing completed and ICT is not empty
PM_CMPLU_STALL_THRD	Completion stalled because the thread was blocked
PM_1PLUS_PPC_CMPL	One or more PPC instructions finished
PM_NTC_ISSUE_HOLD	NTC instruction is held in the issue
PM_ICT_NOSLOT_CYC	Number of cycles the ICT has no itags assigned to this thread
PM_CMPLU_STALL_LSU	Completion stalled by an LSU instruction
PM_CMPLU_STALL_EXEC_UNIT	Completion stall due to execution units (FXU/VSU/CRU)

5.2 Methodology

To evaluate the performance and characterize the applications we use the methodology described in this section. More precisely, we show the system (*5.2.1*) and the tools/software (*5.2.2*) we use for our work.

5.2.1 System in use

Figure 5.3 presents the system employed for this work. It is an IBM Power9 AC992 with 22-cores SMT4; more details are in *Table 5.2*. We include as a competitor to NMC an NVIDIA V100, one of the latest GPU with 32GB of HBM2 memory at 900 GB/s. Indeed, this GPU, similarly to literature NMC works, employs HBM2 memory.
As NMC systems we use a custom hardware design called `Access Processor` (`AP`) [153], which can be mapped on different FPGAs (DDR4 and HBM2). The Access Processor is a custom FPGA design, owned by IBM, that embed NMC concepts

that promises high performance compared to state-of-the art architectures such as CPUs and GPUs.

Table 5.2: *Specifications of the systems employed for the acceleration comparison.*

Architecture	**Configuration**
IBM® Power9 AC922	@3.8 GHz, 22 cores (4-way SMT), 2 sockets, 32 KB L1 cache per core, 256 KB L2 cache per core, 120 MB L3 cache per chip, 512 GB DDR4 2666 MHz
NVIDIA V100-SXM2-32G	@1.53 GHz, 640 Tensor Cores, 5120 NVIDIA CUDA® Cores, NVlink interconnect 300 GB/s 32 GB HBM2 at 900 GB/s
AlphaData 9V3 FPGA	788 FFs, 394k LUTs, 2280 DSPs, 25.3 Mb BRAM, 90.0 Mb URAM, 8 GB DDR4 2400 MHz
AlphaData 9H7 FPGA	2607 k FFs, 1304 k LUTs, 9024 DSPs, 70.9 Mb BRAM, 270 Mb URAM, 8 GB HBM2 at 460 GB/s

Differently from a classical general-purpose computer, where the access bandwidth and latency depend on a complex mixture of workload characteristics and the memory hierarchy, the `Access Processor` (`AP`) design comprises the so-called memory controller, which has the feature of enabling more control over the memory system and programming all the concurrently running data streams from/to the attached NMC accelerators (see *Fig 5.3*). The Access Processor may be advantageous compared to state-of-the-art architectures, such as CPUs and GPUs, for specific memory-bound workloads because it optimizes memory transfers avoiding cache hierarchies by directly accessing the main memory.

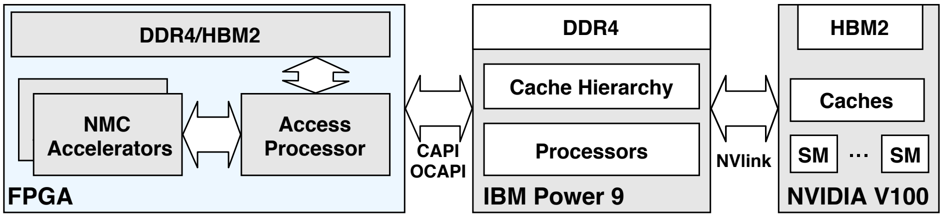


Figure 5.3: *System employed [153]. The figure highlights the host processor, an IBM Power9 with DDR4 memory, that is connected to an NVIDIA V100 with NVLink and to an FPGA with CAPI/OpenCAPI. The near-memory computing (NMC) accelerator prototypes are deployed on the FPGA.*

The key features of the `AP` (see *Figure 5.4*) are: 1) the B-FSM, a programmable state machine technology, applied successfully to a wide range of co-processor devices [154]; 2) programmable address mapping scheme that can highly optimize the bandwidth utilization reducing bank conflicts and managing the data organization.

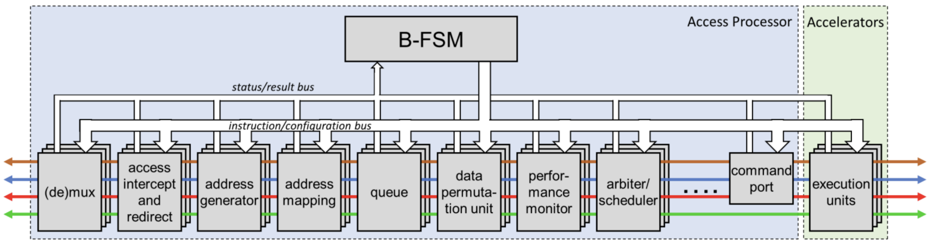


Figure 5.4: *Detailed schematic of the Access Processor design. It mainly consists in the B-FSM, a state machine. The AP provides several units for different tasks. Indeed, many of them are employed to efficiently manage data (address generator and mapping, access intercept and redirect) and to schedule accelerators (arbiter/scheduler).*

2D FFT acceleration on `AP` is performed as a combination of multiple 1D FFTs and transpose (see *Figure 5.5*). It consists of performing a 1D FFT over all the rows of the image and an on-the-fly partial matrix transpose. Then, a 1D FFT is performed on the transposed columns of the images and they are transposed again on-the-fly. In this work we employed performance estimation, which is conservative, for the `AP` based on experiments, e.g running 1D FFTs and matrix transpose.

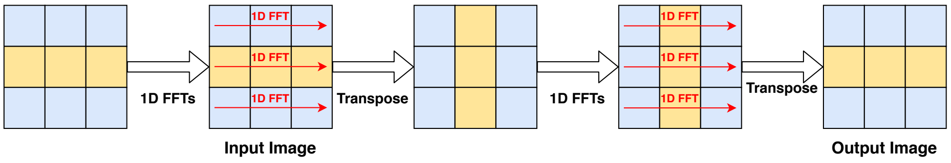


Figure 5.5: *The 2D FFT is decomposed in 1D FFTs: we first apply 1D FFTs over the image's rows and then we transpose it. Finally we repeat the same operation, which results into applying the 1D FFTs over the columns.*

5.2.2 Tools and Software

As a small experiment, testing our tools and analysis methodology, we show in *Figure 5.6* the CPI breakdown analysis (y-axis shows the PMU percentage over the total run cycles) applied to three simple benchmarks: mac, which is a compute-bound kernel written using IBM Power9 intrinsics that perform fused multiplication and accumulate over the same array of data; sgemm, which is a single-precision general matrix to matrix multiplication; and stream-add, a common memory-bound benchmark used to compute the peak bandwidth of a system. We make the following two observations. First, the PMUs not included in the bar chart are nearly 0%, thus showing the relevance of the selected counters. Second, we can distinguish clearly a separation between a kernel completely memory bound (stream-add), which spends most of the time stalling on LSU (load-store units), and another one compute-bound (mac), which spends most of the time stalling on the computing units.

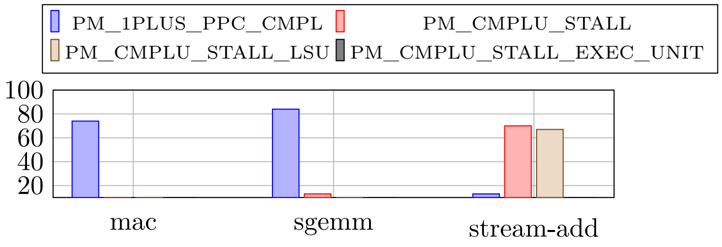


Figure 5.6: *CPI Breakdown applied to micro-benchmarks. Compute bound applications such as a kernel with only multiply and accumulate (mac) instructions and sgemm spent most of the time completing power-pc (PM_1PLUS_PPC_CMPL) instructions. Contrariwise, a stream-add kernel spends most of its time stalling on the load-store units (PM_CMPLU_STALL_LSU).*

FFT was run on CPU using FFTW3 version 3.3.8 and on GPU using cuFFT of the CUDA library version 10.1. Furthermore, we improved the Degridder and Gridder algorithms on Power9 porting the Intel-based code employing IBM Power9 intrinsics. In particular, the main optimization was to use a sine/cosine lookup table, which was implemented with AltiVec intrinsics [155]. Especially this algorithm section of Gridder/Degridder with sine/cosine operations is challenging on other CPU platforms as well; for instance on Intel high performance is obtained employing MKL (math kernel library), which is not available on PowerPC. We use IDG [140] version `5736086c` employing the parameters in *Table 5.3.*

Table 5.3: *Image Domain Gridding parameters.*

Parameters	**Values**
Stations	120
Channels	16-32
Timesteps	8192
Grid Size	4096-8192-16384
Sub-grid Size	32
Cycles	1
Grid Padding	1.0

In order to characterize the application we employed `perf` [73] for extracting the PMUs values.

5.3 Results

We discuss the application characterization results in *Section 5.3.1* and the evaluation of large 2D FFTs on IBM Power9 CPU, NVIDIA V100 GPU and Access Processor prototypes on DDR4- and HBM2-equipped FPGAs *Section5.3.2*.

5.3.1 Application Characterization

We present the CPI breakdown analysis applied to Image Domain Gridding on IBM Power9 in *Figure 5.7*. More precisely, we show the trend of the most interesting performance counters (y-axis shows the PMU, i.e. number of cycles measured, percentage over the total run cycles) on increasing the visibilities grid size. The FFT spends more time on stalling on the load-store units compared to Gridder and Degridder, which means it is more memory bounded. Moreover, FFT spends less time on stalling on the execution units. Furthermore, the FFT becomes increasingly memory bound with larger grid sizes (see 16k vs 8k in *Figure 5.7*) reflecting in a larger time spent on executing it (see *Figure 5.1*).

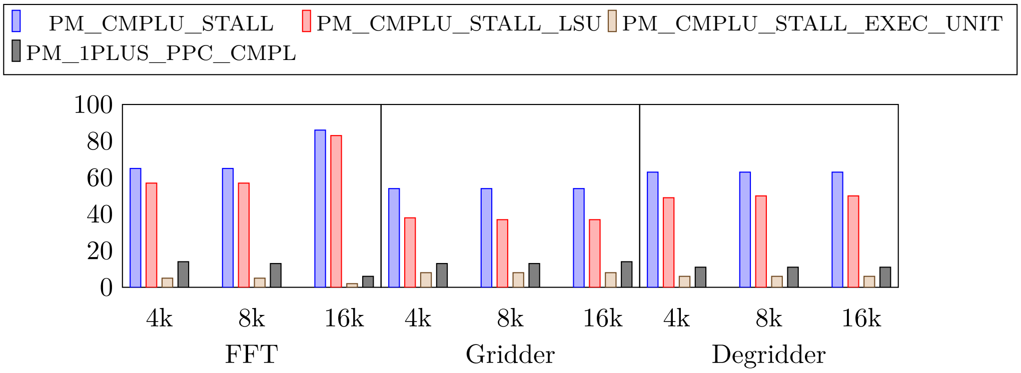
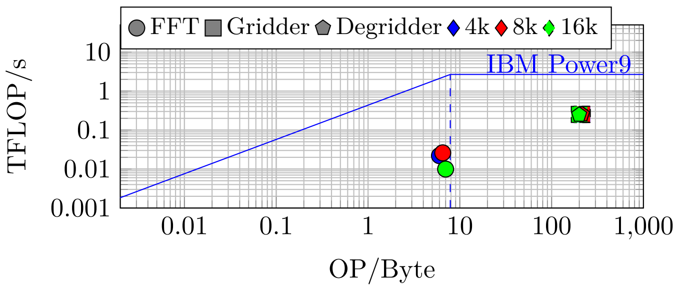


Figure 5.7: *Power9 CPI Breakdown analysis of Image Domain Gridding. The FFT stalls more on Load Store Units (LSU) compared to Gridder and Degridder. Furthermore, it becomes much more memory bound for larges sizes.*

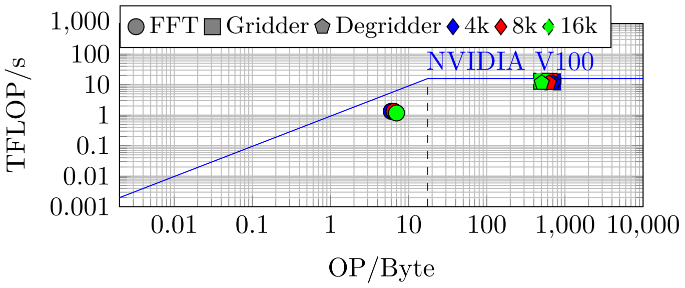
We further analyze the application on IBM Power9 employing the well-known technique of the roofline model [65], see also *Section 2.2.1*. Power9's bandwidth is 340 GB/s for 2 sockets and the peak performance is estimated employing the following formula:

$$TFlops = \frac{freq\ [GHz] * \#_{op.\ per\ core}\ * \#_{cores} * \#_{sockets}}{1000} \tag{5.1}$$

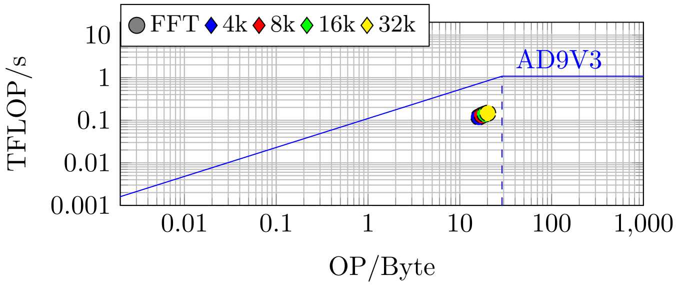
Each core of the IBM Power9 can perform 16 parallel single precision operations. Using the other information from *Table 5.2* we get a peak performance of 2.675 TFlops.
Figure 5.8 shows the roofline model for the kernels in Image Domain Gridding. In particular, we notice that FFT is memory bounded as it is underneath the peak bandwidth ceiling while Gridder and Degridder are compute-bound since they are underneath the peak performance ceiling. Furthermore, the FFT with a grid-size of 16k shows lower performance compared to the FFT performed with smaller grid-sizes. This behavior is due to the larger amount of time spent on stalling in the LSUs. Furthermore, the performance on Power9 remains low compared to the other architecture.
We also include the roofline model of Image Domain Gridding on NVIDIA V100 (see *Figure 5.8b*). Peak performance is reported on the card datasheet (900 GB/s and 15.7 TFlops). On NVIDIA V100 IDG achieves higher performance compared to Power9 for similar kernel characteristics. Furthermore, we build the roofline model for the 2D FFT on `Access Processor` employing the methodology proposed by Intel [156]. More precisely, using this methodology, which estimates the peak performance computing the maximum number of adders that can fit on the FPGA consuming all the DSPs and the logic cells, we compute the peak performance for the two FPGA boards respectively of 1.080 TFlops and 3.675 TFlops. The maximum memory bandwidth is 37.5 GB/s for 2 DDR4 banks at 2400 MHz and 460 GB/s for the HBM2. We show that using the FPGA with DDR4 the FFT reaches higher performance compared to Power9 and it is memory bound (see *Figure 5.8c*). Contrariwise, the higher bandwidth on the FPGA with HBM2 further increases the performance and makes the kernel compute-bound (see *Figure 5.8d*). The arithmetic intensity shown in the roofline models differs for CPUs and GPUs because of the cache effects. Indeed, while the FLOPs are the same, the DRAM traffic can differ. This is not happening for the FPGA prototypes because we compute how many operations (FLOPs) the FPGA performs, and data (Bytes) is moved from/to the main memory, which does not change from the DDR4 to the HBM prototype. Moreover, the FPGA has similar performance compared to GPU having lower peak bandwidth and peak performance. The more efficient use of the memory is shown in *Figure 5.8*, where the arithmetic intensity achieved by the FPGA is higher.



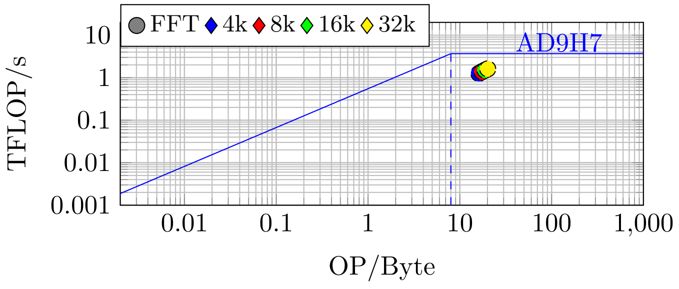
(a) Image Domain Gridding on IBM Power9.



(b) Image Domain Gridding on NVIDIA V100.



(c) 2D FFT on `Access Processor` with DDR4.



(d) 2D FFT on `Access Processor` with HBM2.

Figure 5.8: *Roofline Analysis of Image Domain Gridding on the selected architectures. On IBM Power 9 the FFTs are memory bound, but do not perform near the roof. Contrariwise, on the NVIDIA V100 and the NMC prototypes with DDR4 the performance are close to the memory bound peak. On the NMC HBM prototype the FFT becomes compute bound because of the FPGA's HW characteristics and reaches its performance peak.*

5.3.2 Offloading on NMC Systems

The `AP` provides fine-grained control to schedule the accesses to the DDR4 and HBM2 memory (see *Figure 5.9*), the transfer of the data to and from the FPGAs internal SRAM (Block RAM and/or UltraRAM), and the processing of the data [153]. Because the various 1D FFTs (see *Figure 5.9*) are calculated in parallel using multiple accelerators (the 1D FFTs design used is taken from [157]), the `AP` can schedule the transfer of the input data for each 1D FFT computation from a DDR4 DIMM or HBM2 memory channel to a given accelerator during the time that additional 1D FFTs are being computed on the other accelerators. Therefore, the data communication can be totally hidden.
The same applies to the transfer of the 1D FFT results from an accelerator back to the DDR4 or HBM2 memory. As a result, the access, transfer, and processing of the input data and results for the 1D FFT calculations on the rows of the matrix can be overlapped in an almost seamless fashion, which enables to obtain very high performance by achieving near-optimal utilization of the available DDR4 or HBM2 memory bandwidth [158]. In this case, the 2D FFT performance will be determined almost entirely by the available memory bandwidth, on the condition that there are enough accelerators available to fully overlap the memory access and transfer times. Experiments with FPGA cards that include DDR4 [159] and HBM2 [160] memory have been used to validate this statement.

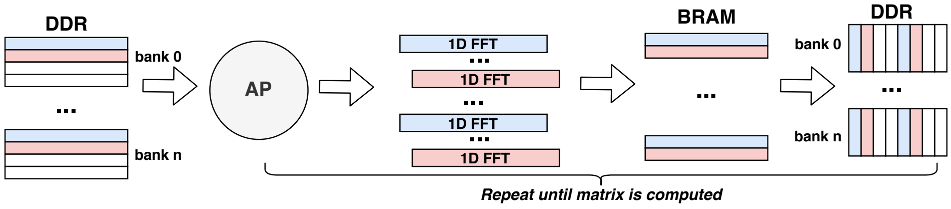


Figure 5.9: *Representation of the design with a focus on the data layout employed for a 1D FFT over the rows plus a transposition on the fly. This represented operation is done twice to perform a 2D FFT. The* `Access Processor` *reads the data from the DDR and then schedules the image rows on different 1D FFT accelerators overlapping their executions. The data is then stored into BRAM, ensuring the row numbers are equal to the data width of the DDR. Thus making it possible to make an on-the-fly transpose before storing the data back to DDR. The key feature of the* `AP` *is the capability to read from different memory channels, if available, concurrently and overlap the data distribution and execution on different accelerator instances, in this case, 1D FFTs.*

By temporarily storing the 1D FFT results for k consecutive rows in internal memory (e.g., Block RAM), with k being equal to the number of samples fitting within the access width of the DDR4 DIMM or HBM2 memory channel (e.g., k=4 64-bit samples would fit in a 256-bit wide access vector to the HBM2 memory), the transpose can be performed on the fly when writing the k row 1D FFT results back to the DDR4 or HBM2 memory (*Figure 5.9* shows this procedure). The same operation as described above is then repeated for the columns to obtain the overall 2D FFT results over the matrix. The effective memory access bandwidth is measured to be equal to 15 GB/s for a single DDR4 DIMM and 10 GB/s for a single HBM gen2 channel (which are conservative values also including the estimated impact of refresh operations, FPGA speed limitations, etc.), then the following execution times can be derived for the computation of the following four differently sized 2D FFTs using DDR4 and HBM2 memory:

Table 5.4: *Estimated execution time of Access Processor for a single 2D FFT.*

Size	**1 DDR4 DIMM**	**2 DDR4 DIMM**	**1 HBM2 channel**	**32 HBM2 channels**
	15 GB/s	30 GB/s	10 GB/s	320 GB/s
4 k	0.033 s	0.017 s	0.05 s	0.0016 s
8 k	0.13 s	0.067 s	0.20 s	0.0063 s
16 k	0.53 s	0.27 s	0.80 s	0.025 s
32 k	2.1 s	1.1 s	3.2 s	0.10 s

As shown in *Figure 5.1* 2D FFT is the main bottleneck in IDG when enlarging the image size. We evaluate the benefits of applying NMC to FFT and comparing it to a Von-Neumann architecture. More precisely, we offload the 2D FFTs and their inverses to the AP design and to the NVIDIA V100.
We show how a NMC approach can be faster than a common CPU (see *Figure 5.10*) outperforming it up to 120x. The proposed design can reach similar performance compared to a high-end GPU using less memory and having a maximum bandwidth lower than half. Furthermore, the FPGA has a lower thermal design power (TDP) compared to CPU and GPU, as reported in the data-sheets[1,2], which is 25W for the DDR4 board and 150W for the HBM2 board. Indeed, the used IBM Power9 consumes around 480 W when performing the FFT and the NVIDIA V100 around 170 W. We extract the power consumption with AMESTER[2] tool on the Power9 system including the NVIDIA V100. Thus making FPGAs good candidates for accelerating radio-astronomy applications.

5.4 Related work

In this section we provide the related work on workload characterization (*5.4.1*) and on the acceleration of 2D FFT kernels (*5.4.2*).

5.4.1 Application Characterization

A large amount of research has been focused on how to characterize workloads to detect bottlenecks. Yasin et al. [96] presented a similar approach to the one

[2]https://github.com/open-power/amester

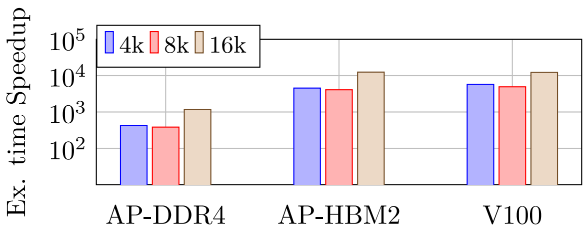


Figure 5.10: *NMC-based platform and NVIDIA V100 execution time speedup compared to IBM Power9. Memory bandwidth is a key factor to get high performance on FFT-based applications. Indeed, the NVIDIA V100 and the NMC HBM prototype achieve much higher performance. The NMC system is close to the GPU performance.*

used in this work, but on Intel systems being the foundations of the well-known Intel VTune [64]. It consists of a top-down approach to identify architectural bottlenecks using selected PMUs. Awan et al. [126, 161] use that approach to spot architectural bottlenecks in big data applications. Differently, other approaches have been studied to characterize the application to be independent of the hardware. Corda et al. [30, 31] analyzed application at LLVM-IR level to extract intrinsic application features focusing on NMC (see *Chapter 3*). However, PMUs are faster to be used and more accurate. As side-effects PMUs are strictly dependent on the HW employed.

5.4.2 Large 2D FFT acceleration

Fast Fourier Transform is one of the most widely studied algorithms in the past. Especially, large 2D FFTs that are expensive on CPU, because of the enormous amount of data that must be moved from main memory through the cache hierarchy and vice-versa, have been improved. Dang et al. [162] proposed an FFT implementation on GPU clusters applied to large electromagnetic problems. Yu et al. [163] and Akin et al. [164] developed two tiling algorithms to improve performance on the 2D FFTs on different platforms. Differently from the previous work, we employed a new computational paradigm called near-memory computing and we focused on larger 2D FFT sizes applied to radio-astronomy imaging.

5.5 Summary and conclusions

We analyzed the state-of-the-art gridding and degridding imaging algorithm for radio-astronomy, as used in SKA, the largest radio telescope on Earth. We employed the CPI breakdown analysis and the roofline model on IBM Power9 identifying the memory bottlenecks. Then, we showed how these bottlenecks can be alleviated by applying an NMC approach to FPGA and comparing it to Power9 CPU and V100 GPU. Thus showing how an NMC approach can highly outperform a CPU and can achieve similar performance compared to a high-end GPU, which has higher memory bandwidth and memory size.

Limitations: although NMC seems a promising alternative to common state-of-the-art solutions, this is not entirely valid for specific real-world applications. Indeed, NMC prototypes are not widely available, leading to the use of software simulators or emulation on FPGAs. These solutions require a great effort to be deployed and, in this case, show limited advantages. Nowadays, as shown in the *Chapter 7*, the upcoming generation of state-of-the-art hardware such as CPUs and GPUs will embed solutions typically employed in NMC such as 2.5D/3D stacked memory and RISC cores.

6 Reduced-Precision Acceleration of Radio-Astronomical Imaging on Reconfigurable Hardware

Radio telescopes produce large volumes of data that need to be processed to obtain high-resolution sky images. This is a complex task that requires computing systems that provide both high performance and high energy efficiency. Hardware accelerators such as GPUs (Graphics Processing Units) and FPGAs (Field Programmable Gate Arrays) can provide these two features and are thus an appealing option for this application. Most HPC (High-Performance Computing) systems operate in double precision (64-bit) or in single precision (32-bit), and radio-astronomical imaging is no exception. With reduced precision computing, smaller data types (e.g., 16-bit) are used to improve energy efficiency and throughput performance in noise-tolerant applications. We demonstrate that reduced precision can also be used to produce high-quality sky images. To this end, we analyze the gridding component (Image-Domain Gridding) of the widely-used WSClean imaging application. Gridding is typically the most time-consuming step in the imaging process, differently from 2D FTT that becomes a bottleneck with high-resolution images only (see *Chapter 5.* Therefore Gridding is an excellent candidate for acceleration. We identify the minimum required

This chapter is based on: S. Corda et al., "Reduced-Precision Acceleration of Radio-Astronomical Imaging on Reconfigurable Hardware", IEEE Access 2022.

exponent and mantissa bits for a custom floating-point data type. Then, we propose the first custom floating-point accelerator on a Xilinx Alveo U50 FPGA using High-Level Synthesis. Our reduced-precision implementation improves the throughput and energy efficiency of respectively 1.84x and 2.03x compared to the single-precision floating-point baseline on the same FPGA. Our solution is also 2.12x faster and 3.46x more energy-efficient than an Intel i9 9900k CPU (Central Processing Unit) and manages to keep up in throughput with an AMD RX 550 GPU.

The chapter starts with background information on reduced precision in *Section 6.1* and the employed methodology in *Section 6.2.* Then, it proceeds with the WSClean characterization in *Section 6.3.* The custom accelerator on FPGA is shown in *Section 6.4* and it is evaluated in *Section 6.5.* Finally, this chapter concludes with related work in *Section 6.6* and conclusions in *Section 6.7.*

6.1 Reduced precision

Reduced precision is a software and hardware technique employing smaller data types to improve performance and energy efficiency. It can be applied at the software level if the architecture supports reduced data types, e.g. half precision in modern GPUs. However, a custom architecture should be designed on FPGA (or ASIC) hardware when a non-supported data type is needed. The main benefits of this technique are the reduced processing elements (PEs) size, which leads to higher throughput, reduce energy and memory requirements, which increase the effective memory bandwidth. Key factors for reduced precision are data types that are described in *Section 6.1.1.* In *Section 6.1.2* we briefly introduce reduced precision and the advent of appealing tools for exploring custom data types in software and hardware.

6.1.1 Data types

Standard architectures, such as CPUs and GPUs, typically support single-precision and double-precision floating-point applications that perform scientific computations. Commonly, single precision is the most widespread data type since the major part of systems supports it. Indeed, even if double precision is supported on most GPUs, they often do not have dedicated units for double-precision computations except for high-end GPUs like Tesla V100 or Ampere A100 [13], thus reducing performance [165]. Radio-astronomical imaging runs precisely enough using single-precision floating-point. For this reason, we focus on data types up to 32 bits. We present in *Figure 6.1* the most commonly used data types today. In particular, we recognize two main categories: standard data types where the bit length is fixed and custom data types where the data length is defined at design time, compile time, or runtime.

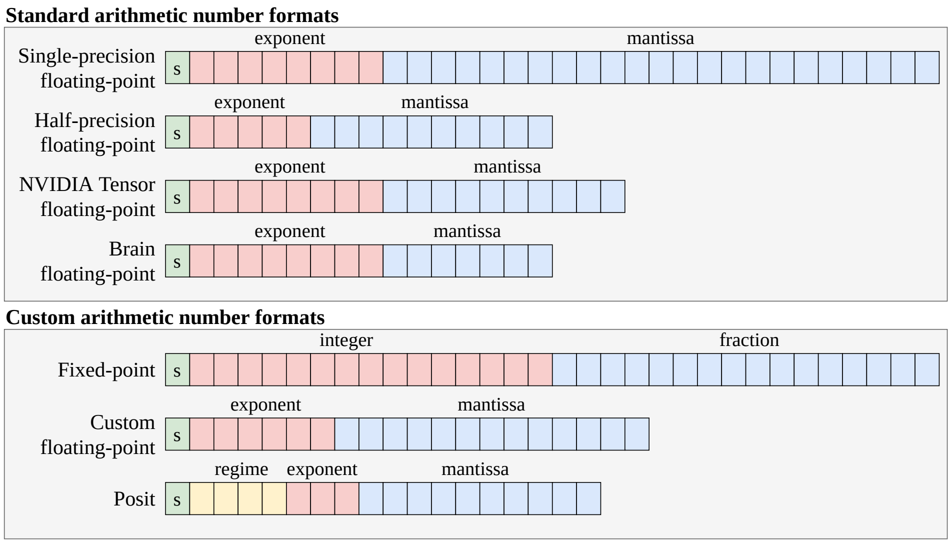

Standard arithmetic number formats
Single-precision floating-point
s
exponent
mantissa
Half-precision floating-point
NVIDIA Tensor floating-point
Brain floating-point
Custom arithmetic number formats
Fixed-point
integer
fraction
Custom floating-point
Posit
regime

Figure 6.1: *Data-types overview. Standard arithmetic number formats such as single-precision and half-precision floating-point are data types usually supported by modern CPUs and GPUs. Custom arithmetic number formats comprise some of the main data types employed in research and are often deployed on FPGAs. The custom formats reported are examples, and the number of bits may differ.*

Except for posit and fixed points, the other data types are floating-point representations that can be expressed by *Equation 6.1.* More precisely, the `exponent` and the `mantissa` bits are responsible for respectively the dynamic range and the precision of the data type. The dynamic range limits the smallest and largest number representable, while the precision is the represented number's resolution (number of digits).

$$number_{floating-point} = (-1)^{sign} * mantissa * 2^{exponent} \tag{6.1}$$

Table 6.1: *Number of bits for the mantissa and exponent of standard floating-point formats.*

Name	**Exponent**	**Mantissa**
Single-precision	8	23
Half-precision	5	10
NVIDIA-Tensor	8	10
Brain	8	7

In *Table 6.1* we present the mantissa and exponent sizes of the main standard data types. **Single-Precision** [166] (or `binary32` [167]) and **Double-Precision** (or `binary64`) **Floating Point** are commonly supported on GPUs and have been added to the **IEEE 754** standard. With the advent of deep learning applications and their noise tolerance and need for reduced length data types, half precision usually offers 2x the performance of single precision in applications that can tolerate the noise introduced by the lower dynamic range and less precision. For the same reason, NVIDIA presents the new `Tensor Float-32` with the release of the NVIDIA A100, the new AI and HPC flagship GPU. This format has the same dynamic range of the **binary32** but reduced precision, which is claimed to be sufficient for most of today's AI applications. **Brain Floating Point** [168] offers a further precision reduction while keeping the same dynamic range offered by single precision. This is especially employed by Intel [169] and Google [170].
Apart from the standard data types mentioned above, other data types, except for several integers (4, 8, 16, 32 bits) in specific GPUs, are not available in mainstream architecture or do not have a pre-defined number of bits. These data types are described below:
Fixed Point [171]: this format is represented by *Eq. 6.2*. A real number is represented by two numbers, one for the integer part and one for the fractional part. Compared to floating point, it has a smaller dynamic range since there is no exponent. Still, the hardware implementation is easier since it considers two numbers (integer and fractional), and it is usually employed on custom accelerators, and FPGA [172, 173].

$$number_{fixed-point} = (-1)^{sign} * integer.fractional \tag{6.2}$$

Custom Floating Point [174]: a custom floating-point is super-set of the floating-point above mentioned. It consists of all the possible floating-point format combinations. They are typically used in embedded systems, where the numeric representation is customized for the specific application by selecting specific bit lengths for the mantissa and exponent fields.
Posit [175]: is a numeric representation that has been proposed as a substitute for floating-point data types. Usually, posit has a higher dynamic range than the floating-point with the same bit length (see *Figure 6.2*). Moreover, posits have a tapered decimal accuracy (see *Equation 6.3*, where x and y are two numbers with same sign) which means that the decimal accuracy reported in *Figure 6.2* is roughly symmetrical, and the highest precision is achieved for numbers near 1 (the horizontal axis is reporting the base-2 logarithm of the numbers).

$$decimal\_accuracy = -log_{10}(|log_{10}(x/y)|) \tag{6.3}$$

This feature differs from floating points that have an almost constant accuracy across the dynamic range, except for small numbers (left side), and the accuracy suddenly falls off a cliff (right side) to accommodate all the NaN (not a number values). However, we are not considering posits as a possible datatype

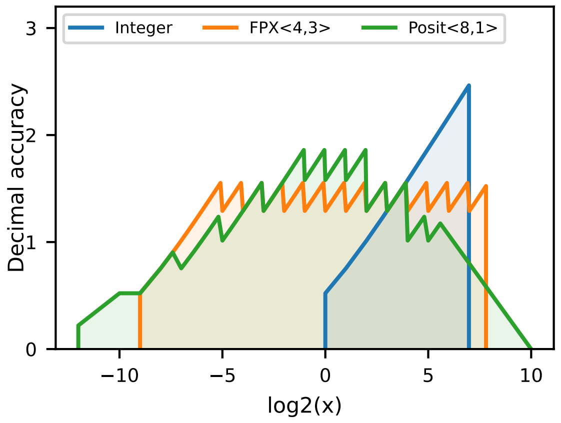


Figure 6.2: *The figure is inspired from [175]. Decimal accuracy for 1) 8-bit signed integers; 2) a custom floating points format with 4-bit exponent and 3-bit mantissa; 3) 8-bit posits with 1-bit exponent [176, 177]. Posits have a larger dynamic range compared to floating points and integers (very small). Integers have higher decimal accuracy for larger numbers. While posits have a tapered decimal accuracy, floating points have approximately a constant accuracy across the dynamic range.*

candidate since operations, like multiplication and addition, are more expensive to implement compared to floating-point [178]. For more details refer to [175].
We employ custom floating point for both analysis and hardware design. This numeric representation comprises standard format such as single precision, half precision, NVIDIA Tensor, and brain floating point. Based on the analysis in *Section 6.3* we evaluate the precision requirements of the target application and discuss the why specific data types are not an acceptable solution for our use case.

6.1.2 Reduced precision tools

Reduced precision is a branch of approximate computing, which usually consists of either reducing the bit size of standard data types or employing more efficient custom data types [28]. Recently, there has been the rise of automated/assisted precision tuning tools and emulation libraries to help and improve the selection of custom data types inside applications. However, the major part of the above-mentioned works support only standard data types [179–181] or fixed point [182]. Flegar et al. [174] designed `FloatX`, a C++ template library capable of emulating `custom` floating point, which we employ in our analysis. FloatX also has a reduced execution time overhead compared to the previous library since it employs hardware-supported floating-point types as back-end. Recently, High-Level Synthesis libraries for supporting custom floating-point precision have been

researched. These libraries are easy to use and portable compared to RTL approaches [183–185].
DiCecco et al. [183] propose a custom-precision floating-point library (`CPFP` [186]) for High-Level Synthesis, and they evaluate it on a small convolution neural network. While the custom floating-point IP, implemented on FPGA, requires fewer resources than single precision. The FPGA design has a lower throughput than the CPU. Thomas [184, 185] proposed a more efficient (see *Appendix A*) templatized floating-point library for high-level-synthesis (`THLS` [187]). This library, which we employ in this work, is also templatized and eventually supports heterogenous custom floating-point operations. The proposed solution has similar resource consumption when comparing standard data types and notable resource reduction when employing reduced-precision data types. The two approaches mentioned above are easier and portable for FPGA development than employing custom floating-point IPs similar to FloPoCo [188], which need to be used as black-boxes.

6.2 Methodology

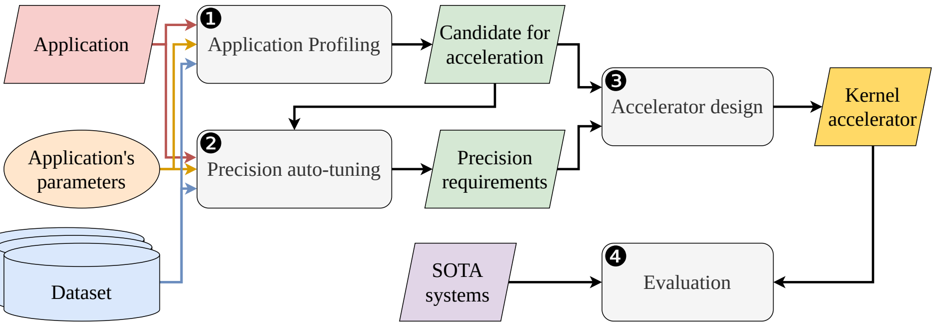


Figure 6.3: *High-level overview of the employed methodology. The application's bottleneck is detected by applying application profiling. Then a precision auto-tuning technique defines the precision requirements for the accelerator, which is designed and deployed. Finally, the accelerator performance is assessed with state-of-the-art (SOTA) architectures.*

We present the methodology employed in *Figure 6.3*. We first ❶ profile the radio-astronomical imager to determine the most time consuming and thus critical kernels. We perform this analysis on different datasets employing the application's parameter described in *Section 6.2.1*. Then, ❷ we evaluate the required minimum precision requirements for the selected kernel using an auto-tuning script based on binary search (see *Section 6.2.2*). Through emulation, it identifies the minimum bit sizes for both the exponent and mantissa for custom floating-point data types, one at a time for the entire kernel. Afterwards, ❸ we perform an accelerator design phase to determine the best design optimization for the selected kernel and the precision requirements (see *Section 6.2.3*). Finally, ❹ we assess the accelerator performance for state-of-the-art systems (see *Section 6.2.4*).

6.2.1 Application Profiling

We profile the most widely used and state-of-the-art radio-astronomical imager, WSClean [189], which also includes the state-of-the-art gridding and degridding algorithm (Image-Domain Gridding) [140], by evaluating the execution time breakdown. We report the software version used in this work in *Table 6.2.*
We employ the `LOFARSCHOOL` dataset, which is usually employed as a test case for practical examples and contains real sky observations [190]. This dataset contains 16 observations and around 30 subbands per observation, available in the LOFAR Long Term Archive (LTA) [191]. We select 14 observations with similar observation parameters such as integration time, observation duration, frequencies, etc. (see *Table 6.3*), and we select the 10th subbands from every dataset; therefore, all the datasets have the same central frequency.

Table 6.2: *Software versions employed. We report the checksum of the commit of the master branch we used for WSClean and IDG. The other packages are WSClean's dependencies.*

Software	**Version**
boost	1.68
OpenBLAS	3.9
python	3.8
wcslib	6.3
cfitsio	3.450
casacore [192]	3.3.0
dysco [193]	1.2
IDG [140]	master 011dfb18
WSClean [189]	master 2680c6a

The WSClean parameters that we use are listed in *Table 6.4.* The sky images are plotted using Kstars FITS Viewer [194] setting the following parameters: shadows 0.0080, midtones 0.0625, and highlights 0.6009.

6.2.2 Precision auto-tuning

To determine the application precision requirements, we employ binary search over the number of mantissa bits, similar to [181], and over the number of exponent bits for custom precision floating-point data, because the execution

Table 6.3: *Datasets observation parameters.*

Name	**Description**
Central Frequency	120.1172-125.7812 MHz
Channels per subband	4
Channel width	48 828.125 Hz
Declination	50.9410-54.8590
Duration	7199 s
Integration interval	2.002 78 s
Right Ascension	311.2500-318.7500

Table 6.4: *WSClean parameters: the top part of the table shows parameters that exclusively depend on the observation and on the radio-telescope structure; the bottom part reports the CLEAN parameters employed for Cotton Schwab run. These parameters are the most commonly used. However, this CLEAN algorithm heavily depends on the user parameters, and a even more complex CLEAN can be used for extracting the sky image such as the Multiscale CLEAN.*

Parameter	**Value**	**Description (unit)**
size	6000 6000	output x and y dimensions (pixels)
scale	5 asec	scale of a pixel (degrees)
use-idg	active	-
auto-threshold	3	CLEAN stop condition (sigma)
niter	50000	number of minor CLEAN iterations
mgain	0.85	gain per major CLEAN iteration
weight	briggs 0	weighting mode and robustness
taper	gaussian 2amin	

time overhead (about 5-10x with respect to the optimized single-precision code) of emulation of custom data types in software makes evaluating the entire search space unpractical.
As shown in *Figure 6.4* we manually instrument the application code to support the software emulation of `custom` data types. To emulate custom floating-point we employ a template header C++ library: FloatX [195] for floating-point [174]. We do not include fixed-point numbers since they would result in very long fixed-point representations; based on the analysis results, which shows that a large dynamic range is required. This binary search algorithm first evaluates the mantissa size and then the exponent size. Since the application usually runs in single precision, the starting mantissa size of 23 bits is divided by two and evaluated. The process consists of first determining the size to evaluate and then updating the headers containing the mantissa and exponent sizes. Then, the application is compiled and run. Finally, the algorithm evaluates the output precision employing the

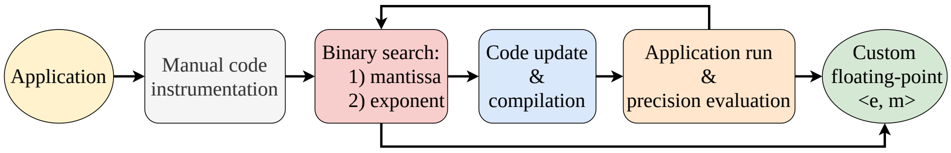

Application
Manual code instrumentation
Binary search:
1) mantissa
2) exponent
Code update & compilation
Application run & precision evaluation
Custom floating-point <e, m>

Figure 6.4: *Binary search algorithm: the application code is manually instrumented to support custom floating-point emulation. We apply binary search first over the mantissa and then over the exponent, because the mantissa mainly determines the accuracy while the exponent is mostly responsible of the dynamic range. The code is updated and re-compiled at each iteration. Then, after the application runs, the algorithm evaluates the output precision based on a threshold. The algorithm provides the representation that should be used for the analyzed code.*

Structural Similarity Index Measure (SSIM) [196] metric.
We select SSIM [196] as the assessment metric since, differently from Peak Signal to Noise Ration (PSNR), we can measure the perceived image quality and thus evaluate how two images are similar. SSIM computation is more complex than PSNR; however, it is computed as:

$$SSIM(x,y) = \frac{(2\mu_x\mu_y + C_1)(2\sigma_{xy} + C_2)}{(\mu_x^2\mu_y^2 + C_1)(\sigma_x^2\sigma_y^2 + C_2)} \tag{6.4}$$

SSIM evaluates different windows images (x and y) using the window average (μ_x and μ_y), the window variance (σ_x and σ_y), the windows covariance (σ_{xy}). The remaining two variables (C_1 and C_2) are employed to avoid instability when $\mu_x^2 + \mu_y^2$ is close to zero. More precisely, these variables are obtained with the following formula $C_{1/2} = (k_{1_2}L_{1/2})^2$, where L is the pixel-value's dynamic range, and k is a constant (usually $k_1 = 0.01$ and $k_2 = 0.03$).
For completeness, we describe the PSNR formula since we report it alongside SSIM. Like Mean Squared Error (MSE), PSNR is straightforward to compute and has specific physical meaning. PSNR computation is shown in *Equation 6.5*, where MAX_I is the maximum value of the original image and the MSE is calculated with *Equation 6.6*, where I is the original image and K is its approximate version.

$$PSNR = 20 \cdot log_{10}(\frac{MAX_I}{\sqrt{MSE}}) \tag{6.5}$$

$$MSE = \frac{1}{mn}\sum_{i=0}^{m-1}\sum_{j=0}^{n-1}[I(i,j) - K(i,j)]^2 \tag{6.6}$$

The auto-tuning algorithm gives as output the mantissa and exponent sizes that should be used to avoid noticeable precision loss. We use a threshold value of

0.99 for the SSIM in order to have an approximate image as much similar to the original one [197]. We employ the precision tuning method with the overhead of running the application multiple times to detect the target precision. However, we use binary search because it has lower time complexity ($O(log_2 N)$) compared to brute force algorithms such as linear search, which has time complexity in the order of $O(N)$, where N is the search space, which in this case is the sum of the number of mantissa and exponent bits.

6.2.3 Accelerator design

Modern Intel FPGAs such as Intel Arria 10 and Stratix 10 have support for single-precision floating-point operations [198]. Their DSPs can perform Fused-Multiply-Accumulate (FMA), which counts as two floating-point operations. However, when employing fixed-point representation, they usually do not reduce the DSPs usage [199]. Differently, Xilinx FPGAs have smaller DPSs [200], thus causing a larger use of DSPs for floating-point operations, e.g., 3 DSPs for a single multiplication. This property makes Xilinx FPGAs a worse candidate for single-precision application compared to Intel FPGAs, but, at the same time, an interesting candidate for exploring smaller data types that can be mapped on a smaller number of DPSs.
Indeed, we target a Xilinx Alveo U50 [201] for deploying our accelerators. The device mentioned above is a small form factor board with a large number of resources, PCIE3 connection, HBM2 memory, and a TDP of 75W. The Alveo U50 is connected to a host system through a PCIE3 X16 connection as shown in *Figure 6.5.* Modern large FPGAs such as the Alveo U50 are built with multiple Super Logic Regions (SLRs). An SRL is a single FPGA die slice contained in an SSI (Stacked Silicon Interconnect) device [200]. We develop the accelerators into the Xilinx Vitis 2020.2 [202] tool flow, which is shown in *Figure 6.5.* Xilinx Vitis needs a source code with embedded OpenCL API to run on the host processor to schedule and control the execution of the accelerators. A similar approach is followed for the accelerator code: the source code containing HLS pragmas or optimization directives is compiled and linked by the Xilinx Vitis compiler. Unlike Intel, Xilinx FPGAs expose developers to a deeper level of optimization details, e.g., array partitioning and IP implementation with resource constraints. The kernel compilation step consists mainly of transforming the source code into HDL. At this stage, programmers can detect possible optimization opportunities and or code inefficiencies. The linking stage maps the accelerator on the FPGA by employing user configuration directives such as computing units and memory channel connections.
We use THLS [187] for mapping custom floating-point operations on FPGA [184]. Since a well-known lack in the Xilinx Vitis accelerator development with OpenCL is the missing support of arbitrary precision [202] libraries, which are fundamental for fixed-precision computation, small integers, and custom floating-point (THLS is built on top of that), we employ C++ with HLS pragmas.

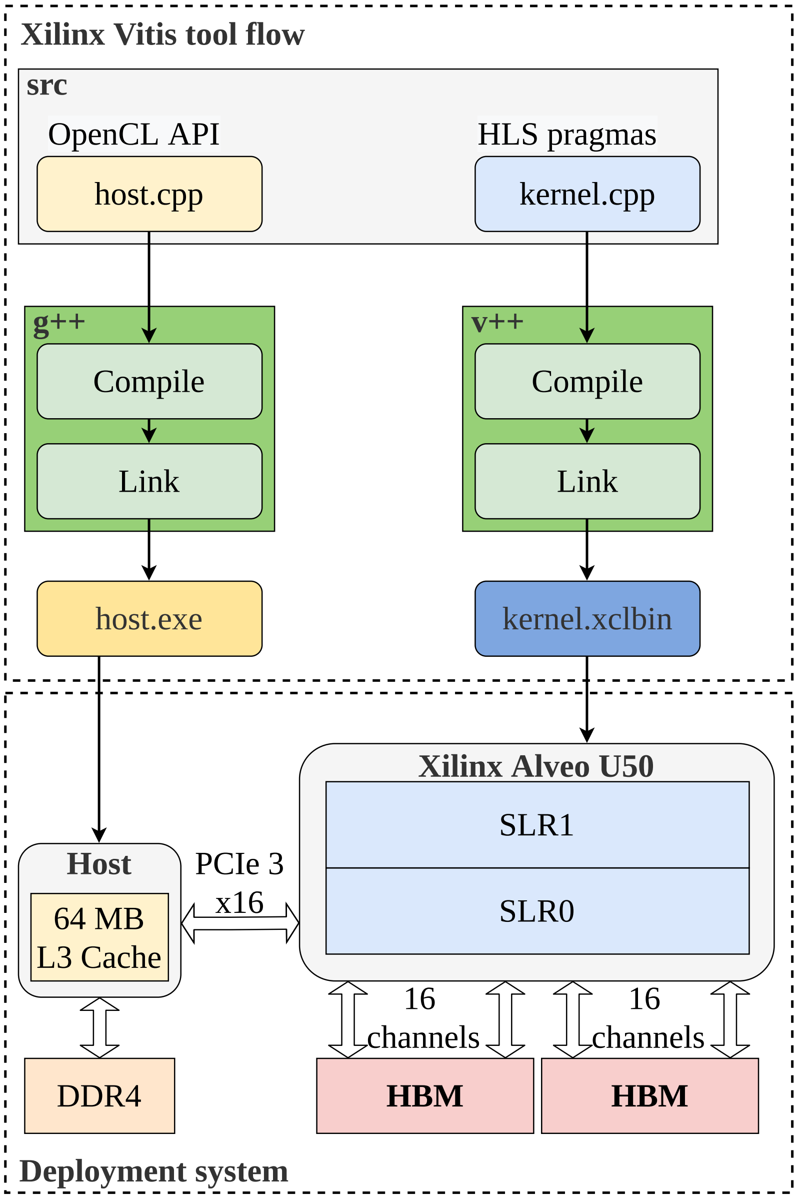


Figure 6.5: *Representation of the Xilinx Vitis toolflow (top box) and how it relates to the deployment system (bottom box). The host code, which runs on the CPU, uses the OpenCL API to communicate with the accelerator on the FPGA and it is compiled with g++. The accelerator code, which runs on the FPGA, includes High-Level Synthesis (HLS) pragmas and is compiled with the Vitis C++ compiler (v++).*

6.2.4 Evaluation with state-of-the-art architectures

Each evaluated architecture needs to be profiled with specific tools. Applications running on a CPU can be profiled with a large number of profiling methods; we choose `perf` [73] since it is available in most Linux distributions and is easy to extract information such as floating-point operations count, DRAM memory traffic, and power consumption (it is usually not needed to have root-access). On the other hand, GPUs typically have proprietary tools. Indeed, we use NVIDIA nvprof [62] and AMD CodeXL [63] for profiling flops and DRAM accesses on the GTX 750 and the RX 550. We count the memory requirements and the number of floating-point operations placed for evaluating the FPGA performance. For measuring the power consumption on FPGA and GPUs, we extend libpowersensor [203]. We select CPU and GPU architectures with similar characteristics such as peak performance and power consumption, which are reported in *Table 6.5.* However, as reported in *Table 6.6*, we have to employ an older NVIDIA GTX 750 instead of an NVIDIA GTX 1050 Ti. This is forced by the missing support of power measurement hardware counters on the GTX 1050/1050 Ti [204].

According to [205], we would expect, for the same chip size, an improvement of ~2x in terms of power consumption efficiency. Furthermore, the TDP values reported for the GPUs are the power-cap limits read in the system out-of-the-box. Indeed, these values are reduced compared to the limits advertised: 35 W instead of 50 W for GTX 750 and 38 W instead of 55 W for RX 550.

Table 6.5: *Hardware configurations of the employed hardware.*

Architecture	**Configuration**
Intel i9 9900k	8 cores, 2 threads per core, 4.0 GHz all cores, 16 MB L3 Cache, 64 GB DDR4 3600 MHz
NVIDIA GTX 750	512 CUDA cores, 1.14 GHz, 2 MB L2 Cache, 2 GB GDDR5
AMD RX 550	8 compute units, 1.09 GHz, 512 KB L2 Cache, 4 GB GDDR5
Xilinx Alveo U50	872 K LUTs, 1743 K Registers, 5952 DPS, 8 GB HBM2

While the peak bandwidths reported in *Table 6.6* are extracted from the device datasheets, the peak performance is computed by multiplying the device frequency and the number of operations that can be computed in parallel in a cycle for each unit. For instance, the Intel i9 9900k has 8 cores that run at maximally 4 GHz; each core can compute 32 flops (we are considering FMA as two flops) per cycle. These values are verified by using synthetic benchmarks such as clpeak [206].

Table 6.6: *Peak performances of the compared architectures. The energy efficiency is the peak estimate derived by the ratio of the peak performance and the the thermal design power (TDP).*

Architecture	**Peak Performance** (TFLOP/s)	**Bandwidth** (GB/s)	**TDP** (W)	**Energy efficiency** (GFLOP/W)	**Process** (nm)
Intel i9 9900k	1.024	57.60	95	10.79	14 Intel
NVIDIA GTX 1050 Ti	2.138	112.1	75	28.50	14 Samsung [207]
NVIDIA GTX 750	1.164	80.19	38	30.63	28 TSMC [208]
AMD RX 550	1.097	96.00	35	31.34	14 GF [209]

Xilinx Alveo U50	**Peak Performance**	**Bandwidth**	**TDP**	**Energy efficiency**	**Process**
Theoretical (724 MHz)	1.547	316	75	19.77	16 TSMC
Theoretical (300 MHz)	0.641	316	75	8.55	16 TSMC
Empirical (292 MHz)	0.535	316	75	6.84	16 TSMC

We apply a similar computation for evaluating the performance of the Alveo U50. As shown in [210], we compute the theoretical performance of the Alveo U50 considering the maximum number of FMA operations (one addition and multiplication [211]) that could be theoretically be placed on this FPGA. More precisely, we count 5 DSPs for each FMA operation, and we consider the available resources as reported in *Table 6.7*. Then this number is multiplied by the maximum of the frequency advertised by Xilinx for the single-precision addition and multiplication IP, which is 724 MHz [211]. A theoretical peak of 1.547 TFLOP/s is obtained by multiplying this number by 2 since we consider the FMA as 2 flops. Since it is very difficult to achieve such frequencies, as reported in [210], we employ the one Xilinx advertised for this FPGA: 300 MHz. The resulting performance (0.641 TFLOP/s) is considerably lower with respect to the theoretical one. However, this is still far from what can be realistically achieved on this FPGA. Indeed, [210] shows that usually, a better upper bound is represented by employing 70% of the LUTs or 80% of the DSPs. For completeness, we also compute the peak performance using the FER (FPGA Empirical Roofline model) synthetic benchmark [210], and we obtain a value of 0.535 TFLOP/s, which is really close to what we achieve in one of our highly optimized single-precision accelerator prototypes.

6.3 Analysis

To determine the bottleneck in the WSClean imager, we perform a bottleneck analysis (*Section 6.3.1*). Then, we carry out a data types precision analysis (*Section 6.3.2*) to understand the precision requirements for the identified bottleneck to be used for the accelerator design.

6.3.1 Bottleneck analysis

As shown in *Figure 6.6*, we first evaluate the execution time breakdown of the overall imaging pipeline for different datasets. The trend in the execution time breakdown is comparable for all datasets. More precisely, the most time-consuming step is inversion. Indeed, inversion needs to be run two times more than the prediction to compute the PSF and the dirty image. Typically, the deconvolution algorithm is the less critical phase.

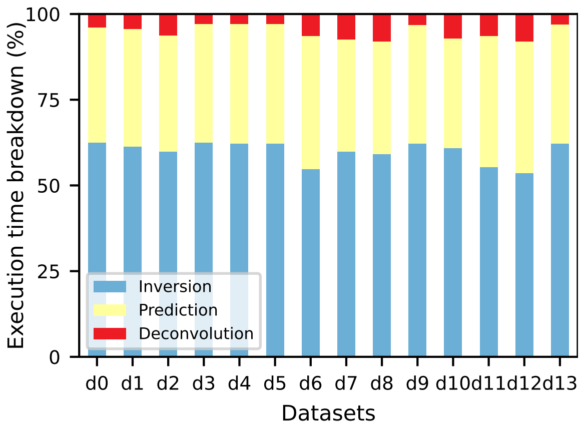
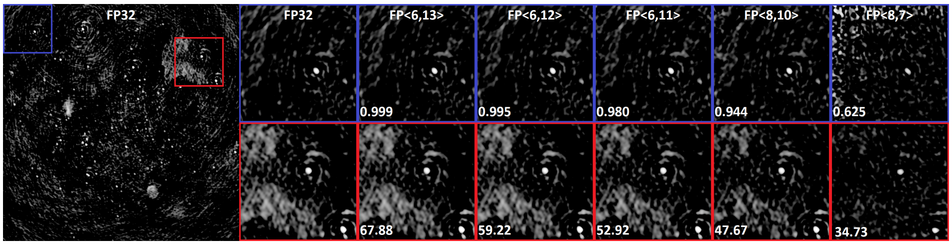


Figure 6.6: *Execution time breakdown of WSClean for different datasets. The most time-consuming phase is the inversion, followed by the prediction. The inversion (light blue) is typically dominated by the execution time of the Gridding kernel. The prediction (yellow) performs the inverse operation compared to the inversion phase; therefore, similar optimization may be applied in the future to optimize the imaging pipeline further.*

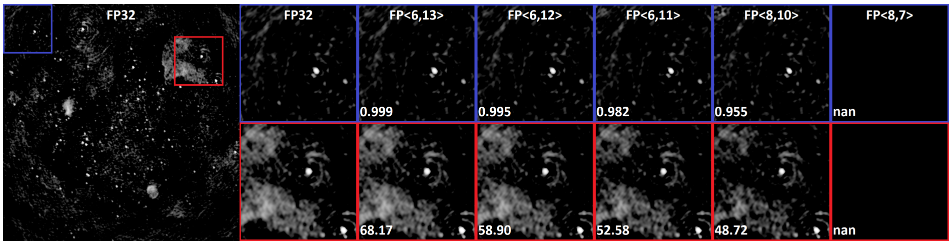
6.3.2 Data types exploration

We evaluate custom precision floating-point data types employing software emulation since common architectures such as CPUs and GPUs usually support single- and half-precision floating points. We notice a fundamental application property: since we are reducing the precision of the gridding kernel, it is sufficient to compare the dirty images instead of the cleaned image to evaluate the accuracy. The algorithm's intrinsic nature easily explains this: the dirty image is a sky image with added noise, and the CLEAN algorithm extracts the brightest sources at each iteration. Thus, each successive iteration needs an equal or smaller dynamic range. This feature helps evaluate the application requirements faster since we need just to run the gridding algorithm once to generate the dirty image, avoiding running the degridding and any CLEAN iterations, thus drastically reducing the analysis time. More precisely, we need to perform the gridding kernel once to

generate the dirty image, once for the PSF, and twice times during the CLEAN major iterations. Therefore, for this particular case, the analysis is about 4x faster than running the whole WSClean application.
We show in *Figure 6.7* how the reduced precision affects the radio-astronomical images. Noise effects can be noticed when using a mantissa of only 10 bits. Another important observation regards the relationship between dirty and clean images.



(a) Dirty images.



(b) Clean images.

Figure 6.7: *Comparison of (a) dirty images and of (b) clean images, both with different data types precision. The numbers enclosed in the angular brackets represent the exponent and mantissa bits. FP<8,10> and FP<8,7> correspond respectively to Nvidia Tensor and Brain floating-point (bfloat) format. While the dirty image a) is a noisy image when using bfloat, the clean image in b) is completely dark. We reported SSIM and PSNR values for the whole images; the blue and red squares are zoomed image sections, which contain respectively the SSIM and PSNR values. We can observe that the SSIM (and PSNR) does not vary significantly between dirty and clean images except for poor quality images. We can notice some visible differences for SSIM lower than 0.99.*

In *Figure 6.7a* the images obtained emulating brain floating point contains a large quantity of noise, the same image cleaned (see *Figure 6.7b*) is empty. Therefore we can conclude that it is sufficient to analyze the dirty image to understand the precision requirements for the entire imager paying attention to the error indicators (SSIM).
In *Figure 6.8* we evaluate the accuracy of dirty images for different data-types for

the gridding kernel in terms of SSIM and PSNR compared to the single-precision floating-point version. The precision requirements depend on the datasets employed, but the combination with 11 bits for the mantissa and 6 bits for the exponent can satisfy almost all the selected datasets. Furthermore, data types such as half-precision and fixed precision are not suitable for this field since their dynamic range is too small. Other data types with very small mantissa, such as brain floating point and NVIDIA Tensor Float, are not accurate enough to correctly represent all the faint features in the image. It is necessary to specify that the Tensor Float representation on NVIDIA GPU can only be used to perform warp matrix-to-matrix multiply and accumulate operations. In this work, we consider this data type representation for the whole kernel considering, e.g., sine/cosine.

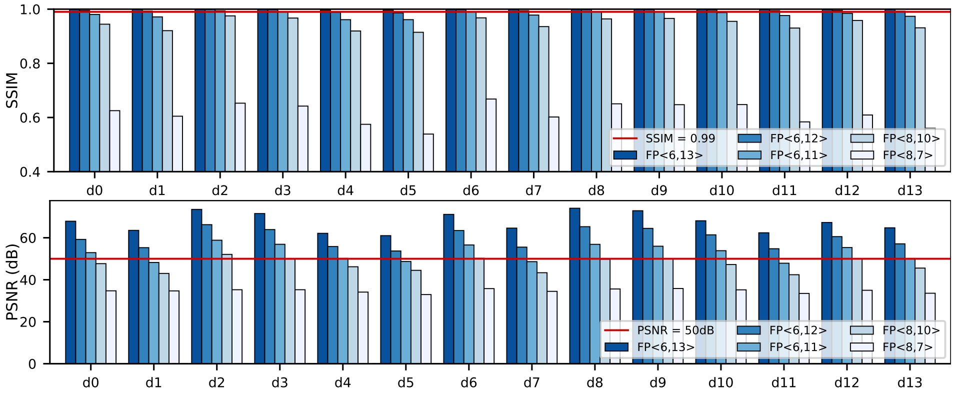


__Figure 6.8:__ Precision accuracy for the selected 14 datasets (d0-d13) for different custom floating-point data types expressed in terms of SSIM and PSNR. The numbers enclosed in the angular brackets represent the exponent and mantissa bits. FP<8,10> and FP<8,7> correspond respectively to Nvidia Tensor and Brain floating point. High-resolution images closely similar to the original usually have an SSIM equal or higher than 0.99 [197]. Given this threshold, reduced precision can produce high-quality images. However, it depends on the datasets. Indeed, in most cases, FP<6,11> or FP<6,12> reach the threshold, while smaller mantissa sizes do not. Such data types are not available on the standard accelerator platform such as GPU, which usually support half, single, double precision. Therefore a custom accelerator is needed. PSNR is just plotted for completeness, and the selected threshold of 50 dB is the maximum value for lossy images, and it serves only as a reference.

6.4 Custom Precision Accelerator Architecture

Optimizing the gridding algorithm on Xilinx FPGAs requires different steps. We first describe in *Section 6.4.1* the high-level structure of the accelerator by explaining the HLS optimizations applied. Then, we explain the employed optimizations for lookup tables and reduced precision respectively in *Section 6.4.2* and in *Section 6.4.3*. Finally, we discuss in *Section 6.4.4* all the different methods we explore for placing the accelerator on the FPGA through Xilinx Vitis and the design points of our prototypes.

Algorithm 6.1: Subgrid computation HLS pseudocode. The core computations are located in line 13 (sine and cosine) and 16 (fused multiply and accumulate). Moreover, the algorithm consists of nested for-loops over radio-astronomical observation parameters such as timesteps, channels and polarizations. The algorithm process the input visibilities employing information regarding the frequencies (wavenumbers), their position in the uv-plane (uvw and uvw_offset), and subgrid pixel parameters (lmn) to compute image' subgrids.

```
   Input: visibilities, wavenumbers, uvw, uvw_offset, lmn
   Result: subgrids
1  subgrids ⟵ 0;
2  for s in subgrids_per_cu do
3      for t in timesteps do
4          for c in channels do
5              #pragma unroll factor = UNROLL_CHANNELS
6              for p in pixels do
7                  #pragma unroll factor = UNROLL_PIXELS
8                  complex<float> pixel[pol]
9                  float lmn [3] ⟵ lmn[p]
10                 float phase_offset ⟵ compute_phase_offset(uvw_offsets,
                    lmn)
11                 float phase_index ⟵ compute_phase_index(uvw, lmn)
12                 float phase ⟵ compute_phase(phase_index,
                    phase_offset, wavenumbers)
13                 float phasor [2] ⟵ cosisin(phase)
14                 for pol in polarizations do
15                     #pragma unroll
16                     pixel[pol] += visibilities[t][c][pol] * phasor
17                 end
18             end
19         end
20     end
21 end
```

6.4.1 Gridding accelerator

We report the HLS optimized pseudocode of the subgrid computation in *Algorithm 6.1* and its high-level representation in *Figure 6.9*. We summarize the main optimizations applied to get our highest performance prototypes:

Memory management: We first optimize the data accesses from the HBM2 memory by using multiple memory channels and widening the AXI (Advanced eXtensible Interface) width to 512 bits for the bus that transfers most of the data. More precisely, we use three channels per compute unit: the first one for the input of the main computation block (subgrid computation), the second one for the input of the post-processing pipeline (Aterms, tapering, reorder, and FFT), and the third one for the output subgrid (see *Figure 6.9*). The data is moved into local buffers to exploit reuse and improve the memory access latency.

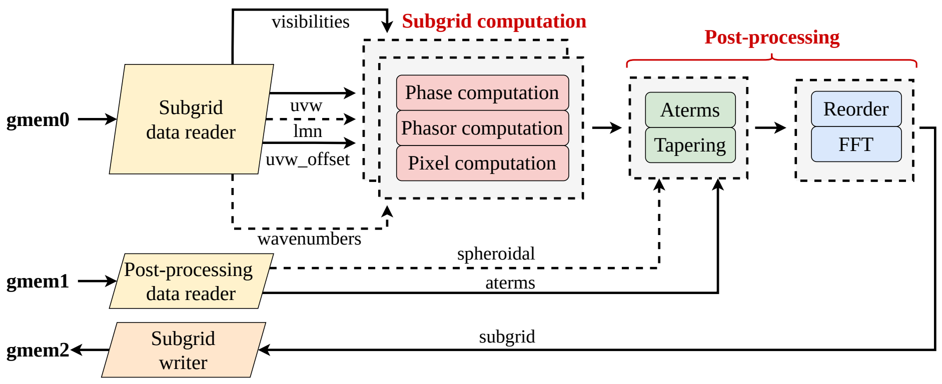
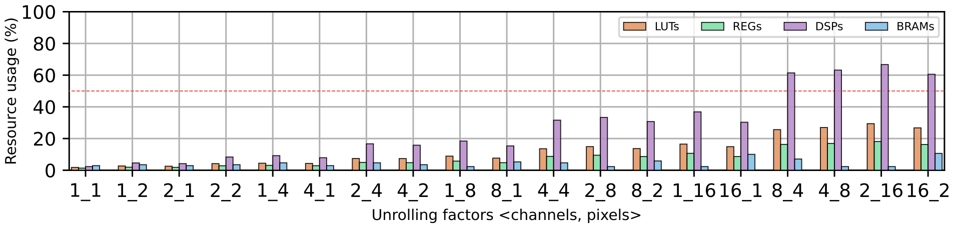


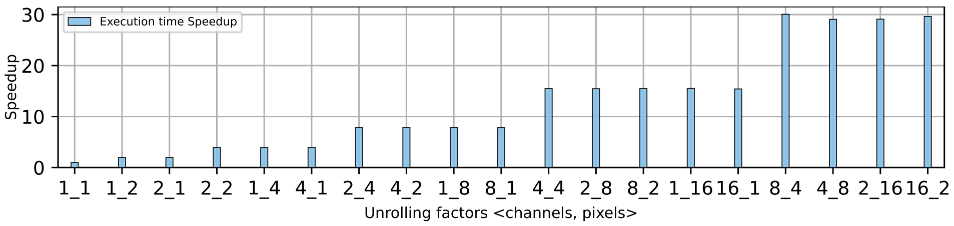
Figure 6.9: *High level scheme of one gridding compute unit: the data are read and write in parallel from different memory channels (gmem); the main computation subgrid computation is replicated N times; finally post processing HW applies the aterms, the tapering and an FFT. Vectorized memory access are reported in black, while the scalar are ones dashed.*

Initiation interval of the subgrid computation: The accelerator described in HLS is implemented as a hardware pipeline where ideally, the pipeline is *stall-free*, and new data is fed into the pipeline every cycle. In this case, the *initiation interval* (II) is equal to 1. In the case of stalls, the II could be larger than 1. Depending on the design, it then takes several cycles for the operations on that data to complete. To achieve II=1, we exchange the loop order of the timesteps and pixels (see *Algorithm 6.1*). The new loop order reduced the Read after Write (RaW) dependencies relative to the subgrid pixel update.

Parallelism: We increase the parallelism with respect to the code mentioned above at different levels. We first increase the parallelism of the subgrid computation by unrolling the channels and the pixels loops. The unrolling is implemented



(a) Resource usage.



(b) Speedup.

Figure 6.10: *Resource usage and speedup of the subgrid computation (not including the post-processing pipeline) for different channels and pixels unrolling factors. The resource values refer to the kernel only, without considering the overhead used by Xilinx Vitis, and are normalized by the maximum number of resources available in the device. The execution time is normalized by the largest values, which is the subgrid computation without unrolling (<1,1>).*

by employing larger local memories (BRAMs) and by unrolling the loops (see lines 5 and 7 in *Algorithm 6.1*) to increase the number of parallel floating-point units. Moreover, *Figure 6.10* reports the performance and the resource usage for different unrolling factors for channels and pixels. While the performance increases almost linearly, resources occupancy makes larger unrolling even more efficient in terms of resource savings, e.g. especially for DSPs (see *Figure 6.10*). Indeed, unrolling the loops of a factor 2 do not imply the 2x more utilization of this resource. However, in order to be able to place more units and have a better area usage and placement, the parallelism should not be excessive, e.g., the case 4_8 is using more than 50% of resources, leaving no space for placing multiple units. Another significant observation concerns the relationship between unrolling channels over pixels: unrolling over the channels increases the BRAM usage and reduces the DSPs usage, which is the opposite behavior obtained by unrolling more over the pixels. This happens because unrolling over the pixels introduces more cosine and sine computation, which requires a larger amount of DSPs than other operations such as additions and multiplications. We conclude that the best trade-off is to have similar unrolling factors for channels and pixels to achieve balanced use of DSPs and BRAMs.

Then, we instantiate multiple subgrid computations (see *Figure 6.9*) that are run in parallel by using the `DATAFLOW` pragma [212]. Finally, we increase the parallelism through increasing the number of instantiated kernel units depending on the design time closure difficulty.

Post processing trade-offs: The post-processing computation consists of applying the A-terms, the tapering, and after a pixel reordering an FFT. As mentioned above, the subgrid computation is replicated `N` times, thus making it possible to have just a single post-processing unit that is capable of processing the data from each subgrid computation in a pipelined fashion. This will add a delay given by the execution time of the post-processing computation, but with the advantage of a constant throughput and reduced resource usage. Since the subgrid computation is much more time consuming compared to the post-processing computation, the latter is designed as cheap as possible to have just sufficient performance to balance the subgrid processing, e.g., initiation interval greater than 1 and low parallelism, and save as many resources as possible, making the subgrid computation the only responsible for the resource mapping on the FPGA board. For completeness, we report that the post-processing section responsible for applying the Aterms and the tapering has an initiation interval of 2 cycles (internal computation, the data is read at II=1 from the subgrid computation), which can be tuned up to II=8 to facilitate the accelerator placing in some instances by reducing resource utilization. We employ this trade off for the single-precision lookup table and custom floating-point design. The section responsible for the FFT has an initiation interval of 3 cycles.

6.4.2 Cosine/sine Lookup Table and reduced precision

Similarly to [104], we employ a lookup table implementation to perform cosine and sine operations and save resources. Indeed, in Xilinx FPGAs, a `cosisin` operation, which computes the cosine and sine of a given angle, requires 11 DSPs compared with the 3 DSPs needed by the lookup implementation. We further reduce the DSPs usage to zero by not representing the phase in radians. We move the multiplication used for the phase conversion in the outer loop and apply it when reading the `lmn` input data, which is a common factor when computing the phase offset and index and consequently the phase. The lookup table implementation consists in saving into BRAMs for pre-computed values for sine and cosine in the range of $[0; \frac{\pi}{4}]$. Then these values are used to compute the sine and cosine by employing the symmetry properties of trigonometric functions. Finally, we have to tune the performance of the post-processing pipeline to let the accelerator achieve better frequencies (about 10 MHz higher).

6.4.3 Reduced precision

The reduced-precision accelerators are obtained by employing the Templatized Floating-Point HLS library [187] for replacing the floating-point operations. Even

if we selected just one numeric format at the time per the whole kernel (homogenous custom floating-point operations), we decide to employ this library instead of the CPFP [186] since it is the most resource-efficient (see *Appendix A*).
In order to have a portable accelerator, we use the same host-kernel interfaces as for the single-precision floating-point accelerators. Then, we add some conversion steps to the input (from single precision to custom precision) and to the output (from custom precision to single precision) of the subgrid computation. This design choice is supported by the fact that the application is purely compute bound, and the conversion does not affect the performance significantly. Since sine and cosine are not available in the library mentioned above, we adapt the lookup implementation used for single-precision by adding a custom floating-point round to integer method that is used to determine the index of the lookup table. As mentioned above, we have to tune the performance of the post-processing pipeline to achieve the accelerator placement at a reasonable frequency (greater than 250 MHz), which means getting a significant speedup. Indeed, when using more resources, the achieved frequency becomes considerably low, e.g., 200 MHz, thus not being beneficial to place multiple units.

6.4.4 Device-specific considerations

After optimizing the high-level synthesis code, one of the main tasks to get the best performance from an FPGA device is the accelerator placement. While it is possible to let the Xilinx Vitis tool flow map the accelerator automatically on the device, it is more efficient in terms of achieved frequency and, consequently performance to fully customize the accelerator mapping. We report the most challenging tasks we face during the FPGA placement:
Super Logic Regions: *Figure 6.11* shows an example of two accelerators with two compute units each placed on the Xilinx Alveo U50. The Xilinx Alveo U50 consists of two Super Logic Regions (SLRs), and it is recommended to map an accelerator on a single SLR. Crossing two SLRs can cause a critical path even if the SLRs are connected with special registers that try to mitigate this problem. During the placement of our accelerators, we find out that each SLR is divided into two subregions [200], and in the middle of them, there is a region consisting of units responsible for managing the input clock. Traversing the two regions as shown in *Figure 6.11* can introduce critical paths that will negatively affect the maximum clock frequency of the accelerator. In our most resource-demanding design, the single-precision lookup table and the reduced-precision ones, we have to instantiate multiple units at different levels to meet timing requirements: 1) one accelerator/compute unit per SLR to avoid SLR crossing, and 2) multiple subgrid computation in each compute units to avoid clock region crossing.
Static region overhead: When mapping application by using HLS a static region (blue area in *Figure 6.11*) is flashed on the FPGA for supporting (accelerated) applications by using HLS (see *Table 6.7*). This static region has the task of managing interconnections such as AXI. It also makes the programmers able

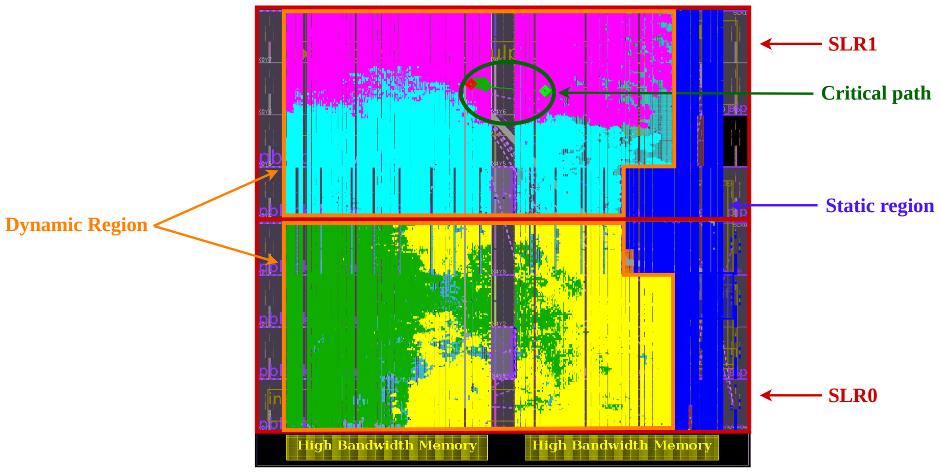


Figure 6.11: *Example of an accelerator placement on Xilinx Alveo U50. In blue is reported the static region for deploying accelerators with Xilinx Vitis. The dynamic regions, which are the areas where the accelerators (yellow, purple, green, light blue) are mapped, are highlighted in orange. The FPGA is divided into multiple Super Logic Regions, FPGA die slices that compose large FPGA boards. Critical paths can usually occur when crossing multiple SLRs. However, critical paths can occur in the same SLR when connecting physically distant components and can be due to traversing the clock region like in the highlighted case.*

to only place the accelerator in the dynamic region, which results, as depicted by *Figure 6.11*, in two asymmetric SLR sub-regions, thus it is important to place, for a large design, more compute units on the left side than on the right side. This observation leads to placing multiple compute elements, whenever possible, in the same accelerator to help the tool find a better hardware placement.

HBM2 memory channels: HBM2 memory can be employed efficiently for both memory-bound and compute-bound applications. In the first case, the large memory bandwidth will improve the application runtime by speeding up the memory transfers. In the second case, which coincides with our case, the multiple channels allow instantiating multiple accelerators that access independent memory spaces. Related to the HBM2 channels, there is another key issue: the Alveo U50 HBM2 memory channels are connected directly to the SLR0 [213], which means that additional logic is needed to transfer the data to the SLR1. When placing multiple accelerators and/or large sub-units that access the memory, connecting the AXI interfaces to distant memory channels is beneficial to avoid a long critical path between memory and processing. We carefully select the placement of HBM2 memory channels to avoid critical paths caused by area congestion, e.g., between two employed HBM2 channels, we decide to leave at least three channels unused.

Table 6.7: *Xilinx Alveo U50 resources: total indicates the overall number of resources of the FPGA, while the dynamic region reports the available resources for accelerator deployment using Xilinx Vitis HLS, which is also reported in percentage.*

Resources types	**Total**	**Dynamic region**	**Available (%)**
LUTs	872 K	731 K	83.83%
REGs	1743 K	1462 K	83.88%
DPSs	5952	5340	89.72%
BRAMs	1344	1128	83.93%
URAMs	640	608	95.00%

More precisely, the Alveo U50 has 32 HBM2 channels, of which only 28 are usable due to power budget limitations (the peak bandwidth of 316 GB/s can be achieved by employing 24 channels) [214]. As shown in *Figure 2.11* we use 3 HBM2 channels per compute unit. Moreover, we employ 6 channels in the lookup and reduced precision implementation because we instantiate two compute units. Furthermore, as shown in *Figure 6.12*, the application is not memory bound for any of the accelerator designs that we considered. Therefore, the bandwidth of a single HBM2 channel is sufficient to satisfy the bandwidth requirements for this application. The choice of using multiple HBM2 channels is determined by the mentioned above congestion area issues.

Vivado strategies [215]: Xilinx Vitis [202] is built on top of Xilinx Vivado, which is responsible for placing and routing the design on the FPGA board. Differently from the CPUs and GPUs programming model, the user can fully customize placement and routing strategies. The choice is between predefined implementation strategies or fully customizable strategies that reduce power consumption, area usage, improved performance, re-timing, etc. However, the main strategies are accessible by the user simply changing the Vitis Compiler optimization flags. We notice that both approaches lead to similar HW results. Indeed, when the frequency of the default Vivado strategy (-O0) is not able to meet the power constraints, we employ the PowerOpt strategy (-O1), e.g., single-precision lookup table, or when it is not able to meet the timing constraint, we use the ExtraTimingOpt (-O3), e.g., to achieve higher frequencies in the reduced-precision accelerators.

Pblock placement: This is a technique that can be applied after the design is implemented for guiding Xilinx Vivado towards a better design placement. With this option, the user can visually select from the GUI where to place certain units by creating a physical constrained region, the so-called pblock. This usually helps in cases of a desired higher frequency. However, we did not notice a significant improvement in our accelerators since they were already close to the highest advertised frequency.

Frequency overclock: Although Xilinx advertises 300 MHz as the maximum frequency for the HLS kernels, in reality, it is possible to achieve higher frequencies. Indeed, in [216] the authors match the same HBM frequency of 450 MHz for small accelerators. However, this is usually not achievable for larger accelerators that usually reach frequency in the range of 200-300 MHz [217]. We manage to increase the frequency of the baseline single-precision floating-point accelerator since the resource usage is not so close to the maximum, and it is able to reach the standard frequency (300 MHz) by just employing the Vivado Default implementation strategy.

6.4.5 Design points

Here we report the main design points of our accelerators:

Single-precision: Our baseline accelerator in single-precision (`FP32`) is obtained by simply using 2 subgrid computations with unrolling factor <4,4> and a single post-processing pipeline. In this case, due to the modest use of LUTs and FFs, it is not necessary to place the computation over different SLRs or apply any particular strategy. We are also able to achieve higher frequencies (`FP32_OC` in *Table 6.8*). Unfortunately, it is impossible to use more resources on this FPGA without violating timings (slow clocks that make the accelerator inefficient) or power constraints (high power budget).

Single-precision lookup-tables: The cosine/sine lookup table implementation can significantly reduce the number of DSPs employed. However, we notice only a small, but still significant reduction of DSPs, since the main computation consists of FMA operations. We manage to improve the performance by placing 50% more computations. This is achieved by using an unroll factor of <2,4> and three subgrid computation units. This accelerator is then instantiated in each SLR, thus having two post-processing pipelines. In this case, to be able to place the accelerator, we have to reduce the number of DSPs by employing the `config_op` option during the HLS compilation to implement all the floating-point additions and multiplication with LUTs. To achieve better frequency, we employ the ExtraTiming_Opt strategy and reduced the post-processing computation's performance. We notice through analysis, a lookup table of 2048 is sufficient to keep the SSIM close to 1. The reported accelerator `FP32_LT` uses a lookup table with 2048 entries.

Reduced-precision: we manage to place 100% more computation with reduced precision compared to the single-precision baseline accelerator and 50% more than the single-precision lookup table implementation. This accelerator consists of 2 subgrid computations with unroll factors <4,4> placed in each SLR. To achieve better frequency, we employ the ExtraTiming_Opt strategy. For the reduced-precision prototypes, we use a lookup table with 2048 entries. In order to achieve the 300 MHz frequency, we employ the reduced performance post-processing computation. Higher frequencies do not meet timing constraints and the power budget (or TDP).

6.5 Evaluation and discussion

In *Section 6.5.1* we report the area usage of the proposed accelerators, and we assess our accelerators performance in *Section 6.5.2* by employing the roofline model [65] and by measuring the throughput and the energy efficiency. Then, we highlight significant lessons learned during this work in *Section 6.5.3*.

6.5.1 Area usage

In *Table 6.8* the area usage of the highest performance accelerators here designed is presented. More precisely, we report as `FP32` the baseline single-precision floating-point accelerator, which can reach the advertised frequency of 300 MHz. Moreover, we show the same design with higher frequency (346 MHz) as `FP32_OC`. Resource usage does not vary significantly. Compared to [104] our DSP usage is significantly lower due to the mentioned above issues regarding the static region and timing closure.

Table 6.8: *Resource utilization for the highest performance Gridding accelerators.*

Version	**LUTs**	**FFs**	**DSPs**	**BRAMs**	**Frequency**
FP32	434 k (49.88%)	604 k (34.72%)	4114 (69.12%)	454 (33.74%)	300 MHz
FP32_OC	435 k (49.99%)	640 k (36.78%)	4114 (69.12%)	454 (33.74%)	346 MHz
FP32_LT	649 k (74.58%)	614 k (35.29%)	3142 (52.71%)	1045 (77.75%)	296 MHz
FPX_6_11	642 k (74.81%)	754 k (43.33%)	1956 (32.81%)	818 (60.86%)	300 MHz
FPX_6_12	656 k (75.39%)	767 k (44.10%)	1956 (32.81%)	818 (60.86%)	300 MHz

The reported single-precision lookup-table (`FP32_LT`) accelerator has a higher overall resource usage because we manage to place more units compared to `FP32`. DSPs usage is lower since the lookup-table sine and cosine computation uses fewer DSPs with regards to the baseline design, and we implement addition and subtraction without DSPs. The achieved frequency of 296 MHz is close to the advertised one, which is difficult to achieve due to power-budget constraints.

The reduced-precision accelerators (`FPX_6_11` and `FPX_6_12`) consume more resources than the baseline, but we can place twice the number of compute units. Furthermore, this design uses fewer resources than the baseline and consumes less power than the design with a single-precision lookup table. As already mentioned `FPX_6_11` consumes slightly fewer resources (LUTs and FFs) than `FPX_6_12` because of the smaller mantissa.

6.5.2 Performance

We assess the performance of our accelerators against CPU and GPUs with similar peak performance and manufacturing technology. We first evaluate the performance achieved by each platform with the roofline model [65, 141] in *Figure 6.12*.

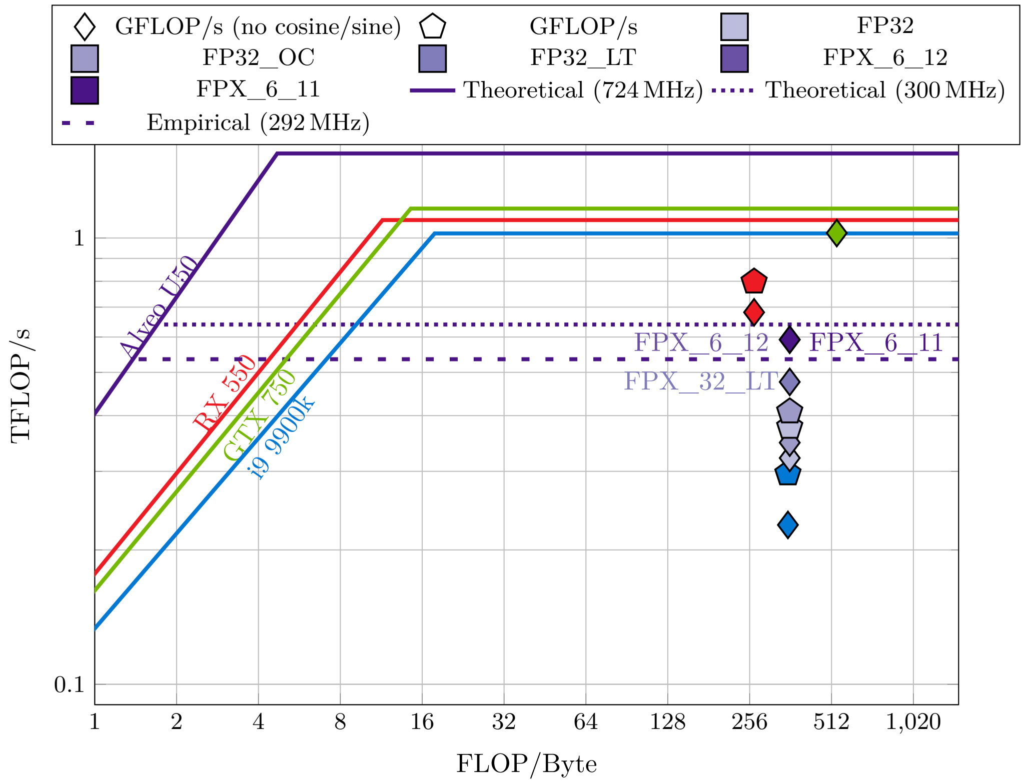
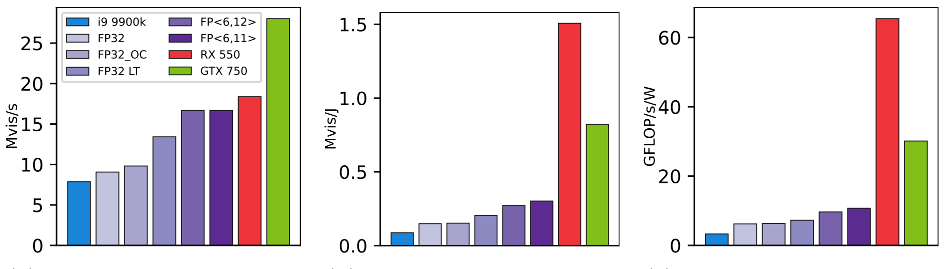


Figure 6.12: *Roofline model of the gridding kernel mapped to different architectures. The pentagon shape represented the TFLOP/s achieved by each platform measured using performance counters. The diamond shape shows the TFLOP/s achieved without including sine and cosine, since specific architectures such as the AMD RX 550, uses multiple floating-point instructions to compute sine and cosine compared to the NVIDIA GTX 750 that has special function units for transcendental math operations [44]. We show only one point for the GTX 750 and the lookup table accelerators (including the custom floating-point prototypes) since the sine and cosine operations are not executed as floating-point operations. We report different horizontal roofs for the Xilinx Alveo U50 based on the discussion regarding peak performance in Section 6.2.4.*

The roofline model shows the performance obtained (TFLOP/s) and the analyzed kernel's arithmetic intensity (FLOP/Byte). The roofline defined in this way is not a measure of throughput but an indicator of how the application can be optimized for a certain architecture. Indeed, we report that only the NVIDIA GTX 750 can reach almost peak performance. This is because cosine and sine operations are offloaded to special units and indeed do not create bottlenecks.
Differently, the AMD RX 550 does not have these special units, and the cosine and sine operations run at a quarter of the speed [218] of single-precision floating-point operations. We further notice that each cosine and sine function requires three floating-point operations. Indeed, in *Figure 6.12* we also show the RX 550 performance numbers taking into account the extra instruction to compute sine and cosine functions. We similarly reported the performance of the proposed accelerators showing how they are performing better than CPU. In particular, our best reduced-precision design is close to the AMD GPU in terms of TFLOP/s. Compared to [104] our baseline design has lower performance due to not having single-precision floating-point DSP support.
Moreover, *Figure 6.12* shows that the application is compute-bound on all the architectures, and its high arithmetic intensity value highlights low memory bandwidth requirements. For instance, considering the RX 550, the roofline shows that the bandwidth required is lower than 3.7 GB/s. Similar results apply to the other architectures in *Figure 6.12*. Therefore, a single HBM2 channel (per compute unit) would satisfy the bandwidth requirements of the application on FPGA. Note, however, that we employed more channels to facilitate timing closure, as previously mentioned. Being compute bound can also be explained at the algorithmic level: the application reads cachable data (e.g., visibilities). It produces the subgrid pixels by performing many operations over this cached data. Thus, the application would be memory bound only in the case of architecture with a lower DRAM bandwidth. Higher bandwidth requirements would also be needed if next-generation FPGAs would enormously increase the computing capability, e.g., with more sophisticated DSPs.
In *Figure 6.13* we evaluate the throughput and the energy efficiency of the accelerators. More precisely, *Figure 6.13a* reports the throughput in terms of Mega Visibilities per second (Mvis). *Figure 6.13a* evaluates the performance of our accelerators. All our designs have higher throughput compared to the i9 9900k up to 2.12x. Our best reduced-precision design is close to the AMD GPU performance. Moreover, the reduced-precision prototype is 1.84x faster than the single-precision baseline. However, the GTX 750 is the faster architecture due to their special function units mentioned above. Our baseline architecture is also outperformed by the Intel counterpart proposed in [104] because of the single-precision DSPs support of Intel FPGAs.
As shown in *Figure 6.13b and 6.13c* our accelerators outperform up to 3.46x in terms of energy-efficiency the CPU. The single-precision lookup table implementation reaches 88.97% of the empirical peak performance. Our best reduced-precision design is 2.03x more energy efficient than the single-precision baseline



(a) Performance in terms of Mvis per second.

(b) Energy efficiency in terms of Mvis per Joule

(c) Energy efficiency in terms of GFLOP/s per Watt.

Figure 6.13: *Performance and energy efficiency evaluation on different architectures. FPGAs prototypes outperform the CPU and are close in terms of throughput to the AMD GPU. However, GPUs show better energy efficiency.*

design. However, it is 78.77% and 63.29% less energy-efficient than the evaluated AMD and NVIDIA GPUs.
We also observe that the AMD RX 550 is more energy-efficient than the NVIDIA GTX 750. This is mainly caused by the different lithography technology. By scaling the performance of the NVIDIA GTX 750 chip to 14 nm [205] the power consumption would be ~2x reduced, thus being more efficient than the AMD counterpart.
Summarizing, our key observations are:

1. Our single-precision floating-point accelerators are more energy-efficient compared to CPUs with similar technology because of the better hardware utilization.

2. Reduced precision improves the accelerator's overall performance being much faster than CPU and achieving comparable performance to AMD GPUs thanks to the higher density of operations that we placed on the FPGA.

3. NVIDIA GPUs reach the highest percentage of peak performance exploited (~88%) compared to the other architectures, especially against AMD GPUs (~73%), thanks to the special units for sine and cosine. Overall, GPUs are the more energy-efficient architecture (see *Figure 6.12* and *6.13*) compared to CPUs and FPGAs with similar features due to their energy-efficient architecture (see *Table 6.6*).

6.5.3 Lessons learned

Radio-astronomical imaging applications usually employ single-precision or double-precision floating-point data types. We evaluate the use of reduced-precision data types for the gridding kernel and make the following observations:

Reduced precision applicability in radio-astronomical imaging: Different from artificial intelligence applications, where it is possible to highly reduce the data size, e.g. 1 or 8 bits [219, 220], radio-astronomical imaging needs higher precision for reconstructing sky images. Indeed, from our analysis, we observe that reduced precision can be applied in a state-of-the-art radio-astronomical imager. However, compared to AI tasks, the required precision is higher to avoid image artifacts. Tensor-Float floating-point numbers have a sufficiently high dynamic range for radio-astronomical imaging kernels, but the number of bits is too low to accurately represent the application values, such as visibilities and subgrids. Moreover, on GPUs this format can only be used for specific warp matrix-to-matrix multiply and accumulate operations.

Benefits of applying reduced-precision in radio-astronomical imaging: Reduced precision is a well-known technique for improving performance and energy efficiency [28]. It is a technique that can be applied to compute-bound applications to reduce the compute unit size and memory-bound applications to decrease memory bandwidth requirements. We observe a significant improvement in GFLOP/s and energy efficiency, respectively 81.96% and 84.71% compared to the standard single-precision accelerator and 25.75% and 33.02% compared to the lookup-table implementation.

FPGAs vs CPUs and GPUs: The state of the art confirms that FPGAs and GPUs are more energy-efficient than CPUs except in rare cases [17]. Most past works shows that FPGAs are more energy-efficient than GPUs. However, these works compare GPUs with higher performance and power consumption. In this work, we show, as presented in [104], that for this particular application domain, GPUs with similar peak performance, thermal design power and manufacturing process are faster and have better energy efficiency. GPUs are also easier to program compared to FPGAs, which require hours of compilation to generate the bitstream. However, FPGAs have the flexibility to synthesize custom data types on hardware for assessing performance improvement against standard data types. In this work, we first evaluate the applicability of reduced precision for radio-astronomical imaging. Then, we design a reduced precision accelerator for radio-astronomical imaging on FPGAs reporting similar performance to GPUs without special sine/cosine units and improved performance and energy efficiency compared to its baseline.

Xilinx Alveo U50: We make the following observations about the Xilinx Alveo U50 FPGA:

- The static region consumes many resources (~17% LUTs and ~13% DSPs) reducing the maximum performance attainable with HLS accelerators.

- The power budget and timing closure constraints, which can be improved with Vivado strategies and overclocking, make it impossible to fit more compute units (see *Table 6.8*) in the designs reaching a maximum of 69% of DSPs (77% counting the static-region effect).

- Large accelerators are difficult to place as discussed in *Section 6.4.4* due to SLR placement. However, as previously discussed, Xilinx FPGAs are appealing candidates due to their DSPs structure (see *Section 6.2.3*).
- A positive feature in HBM-based FPGAs is the possibility to map AXI interfaces to a larger set of memory channels, alleviating congestion issues arising from small numbers of memory channels.

6.6 Related work

Related work on radio-astronomical imaging acceleration on modern computing systems is described in *Section 6.6.1*; moreover, we report related work on accelerating domain-specific-application by using Xilinx Vitis in *Section 6.6.2*.

6.6.1 Radio-astronomy acceleration

Over the past couple of years, hardware technologies have significantly improved, and a fair amount of research has been done on optimizing radio-astronomy algorithms to satisfy the, e.g., the huge Square Kilometre Array requirements [221].
The current state-of-the-art full imager, WSClean, is proposed by Offringa et al. [114]. It consists of an entire radio-astronomical imaging pipeline, including W-stacking, which is an extension of the previous gridding and degridding algorithm, the so-called W-projection, and different CLEAN algorithms such as Högbom, Cotton-Schwab and Multiscale CLEAN [112].
Veenboer et al. [4, 222] optimize the Image-Domain Gridding [105], the current state-of-the-art fastest algorithm for gridding and degridding for radio-astronomical imaging. They show how GPUs could reach almost peak performance and deliver better execution time and energy efficiency than CPUs. Image-Domain Gridding is now part of the WSClean imager. In [104] the authors also accelerate the gridding and degridding kernels on FPGA using a high-level-synthesis methodology based on OpenCL. Their FPGA implementation outperforms CPUs, but the GPU one is more energy efficient. Our single-precision accelerator baseline is outperformed by the one presented in [104] due to the single-precision floating-point DSPs support of Intel Arria FPGA [198]. However, we employ Xilinx FPGAs to assess the performance of a reduced precision accelerator for radio-astronomical imaging.
Hou et al. [223] implement an optimized prototype for the degridding algorithm on FPGA, outperforming both CPU and GPU by respectively 2.74x and 2.03x in terms of energy-delay product (EDP). However, they employ an outdated version of the degridding algorithm called W-projection, which does not reach the performance of IDG and does not include DDE corrections [106].
Corda et al. [33] focus on estimating the benefit of near-memory computing for huge images in IDG. They show that FFT is a critical bottleneck, which

can be alleviated using architectures exploiting High-Bandwidth Memory. More precisely, they demonstrate how an FPGA design could reach a similar performance compared to a GPU with smaller memory and less memory bandwidth (see *Chapter 5*).

Table 6.9: *Radio-astronomy related work.*

Work	Date	Application	Platform	Optimization
Offringa [112, 114]	2014-2017	W-Stacking, CLEAN, (Högbom Cotton-Schwab Multiscale)	CPU	Optimized full imager (WSClean)
Veenboer [4, 222]	2017-2020	Image-Domain Gridding	CPU/GPU	code optimization (added to WSClean)
Grel [224]	2018	Högbom CLEAN	FPGA	Custom accelerator of Högbom CLEAN formulated as a Compressive Sensing problem
Veenboer [104]	2019	Image-Domain Gridding	FPGA	custom accelerator
Seznec [225]	2019	Generic deconvolution	GPU	half-precision deconvolution
Hou [223]	2020	W-Projection	FPGA	custom accelerator
Corda [33]	2020	Large 2D FFT	FPGA	NMC acceleration
This work	2022	WSClean (Image-Domain Gridding)	FPGA	Reduced precision analysis and acceleration

Seznec et al. [225] propose a simple deconvolution GPU implementation using half-precision data type. While half-precision floating-points speed up the algorithm significantly, using this data type causes output degradation depending on the dataset used. Furthermore, they focus on small images (2048x2048 pixels), while radio astronomical images are typically bigger, e.g., 5000x5000, 16000x16000 and larger [4].
Grel et al. [224] formulate the Hogbom Clean as an Iterative Hard Thresholding (IHT), a compressive sensing technique, to reduce the data dimensions. While this work provides compelling insights, they optimize the most simple Clean algorithm and use very low resoluted images (256x256 pixels).
To the best of our knowledge, our work is the first demonstration of a custom floating-point architecture for radio-astronomical imaging by focusing on the main bottleneck (gridding kernel).

6.6.2 Xilinx Vitis

Recent high-level synthesis work on Xilinx FPGAs is based on Xilinx's new programming tool-flow: Xilinx Vitis. Brown et al. [216] show the steps to take when optimizing an application for Xilinx FPGAs by employing Xilinx Vitis. Although some observations in their work have been helpful, they focus on very small FPGA designs (using about 2% of the available resources), while we try to put as many resources to good use as possible. Indeed, for a small accelerator, it

is possible to reach a frequency of 450 MHz, while for realistic use cases, it turns out to be in a range of 200-300 MHz.
Calore et al. [210] benchmark the Alveo U250 with the FPGA Empirical Roofline model (FER), showing that there is a considerable difference between theoretical and attainable performance on FPGA, which usually is not so high in different architectures such as CPUs and GPUs. In our work, we employ the FER benchmark to determine the single-precision horizontal roof for the Alveo U50.
Nguyen et al. [226] evaluate FPGAs from different vendors, comparing them to GPUs showing that modern GPUs are easier to program and have better energy efficient except for rare cases such as fixed-point precision computations.
Choi et al. [227] benchmark HBM-based FPGAs from different vendors to compare the memory performance. They present insightful observations regarding how the HBM2 memory channels must be mapped on Xilinx Vitis, e.g. Alveo FPGAs usually has 30 channels available over a total of 32. Related to this work, we notice that it is crucial to carefully select the HBM2 channel when implementing the accelerator to avoid critical paths that can significantly reduce the achievable frequency.

6.7 Summary and conclusions

We present the first reduced-precision custom floating-point analysis and accelerator for radio-astronomical imaging. More precisely, we evaluate the bottleneck of WSClean, the state-of-the-art radio-astronomical imager. Then, we analyze for the first time the impact of reduced precision on radio-astronomical imaging by employing custom floating-point for the main kernel, the so-called gridding, from the state-of-the-art Image-Domain Gridding algorithm. We demonstrate that reduced precision could be applied to radio-astronomical imaging, but data types must be selected carefully based on the dataset to avoid precision loss. Indeed, our analysis excludes standard low-precision data types supported in modern GPUs and highlights insightful observations regarding how the analysis can be evaluated in a shorter time. We port the gridding algorithm (from Image-Domain Gridding) on Xilinx FPGAs using single-precision floating-point to serve as a baseline. Moreover, we map the first reduced precision prototype of gridding kernel on the same hardware by employing custom floating-point data types. We find that the reduced precision accelerator is up to 1.84x faster and 2.03x more energy-efficient than its single-precision version. Compared to the used CPU, our best accelerator prototype is 2.12x faster and 3.46x more energy-efficient than the CPU. However, it is 9.16% and 40.44% slower and 78.77% and 63.29% less energy-efficient than AMD and NVIDIA GPUs.
Thus we corroborate with the earlier observation that currently GPUs are the most energy-efficient architecture for radio-astronomical imaging, and custom precision support in the next generation of GPUs could further improve the performance/watt for radio astronomy imaging. Even though FPGA design

support improved in the past years, FPGAs would need improved lithography technology [228], more DSPs with improved performance for floating-point and sine/cosine operations, smarter HLS compilers, and faster placement tools to be competitive against GPU in this application domain.
This work paves the way for future research in this specific application domain. Indeed, it would be interesting to explore reduced precision at runtime, e.g., by reconfiguring the accelerators based on dataset requirements and/or user parameters, to further improve performance. Fine-grained reduced precision, which consists of differentiating the data types for different instruction or kernel sections, would be a viable option for higher performance and energy efficiency in future works. The presented work also shows that FPGA technology can be improved to make them more competitive compared to GPU in radio astronomy applications.

Limitations: FPGA technology is an interesting alternative to more traditional architectures such as CPUs and GPUs. Typically FPGAs are faster and more energy efficient than CPUs; however, this is not always true for GPUs. FPGAs can be used to design custom hardware to match application requirements, like in this Chapter. Nonetheless, current FPGA technology can hardly compete with GPUs in applications where high accuracy and high performance are required, such as our radio-astronomical imaging use case.

7 Conclusions

Previous chapters show this thesis work on characterizing and optimizing applications for high-performance. This task is nowadays more difficult with the widespread landscape of heterogeneous architecture available in compute clusters. Therefore, we summarize the findings and the limitations of this thesis' contributions in *Section 7.1*. Then, we describe how the hardware is evolving in the present and near future, describing its main trends (*Section 7.2*); and how these can bring open challenges for future work (*Section 7.3*).

7.1 Summary

Each contribution presented in this thesis pushes forward the state-of-the-art of profiling and optimizing the application for high-performance computing systems. Here, we summarize and discuss the contributions and how they improved the state-of-the-art alongside their limitations.

7.1.1 Application profiling and offloading

Hardware-agnostic application characterization is a viable technique to extract performance-critical metrics from applications without depending on the system on which the profiling is executed. It is essential to evaluate intrinsic application features to understand its bottleneck or design new computing architectures. In *Chapter 3* we provide an extension of the PISA tool with metrics directed towards memory and parallelism. Then, we evaluate the feasibility of this technique in near-memory computing systems, a promising new computational paradigm that shifts the computation from the processing units to the memory.

Limitations: platform-independent analysis can be successfully employed to determine if an application should be offloaded on a accelerator to improve speedup and energy efficiency performance. We mention the case of NAPEL, a high-level framework built on top of PISA using ensemble machine learning. NAPEL can accurately predict unseen application performance on near-memory computing systems. Moreover, this framework can reduce the prediction time since it does not need to run the system simulator during this phase. However, the platform-independent application analysis is still applied to the prediction phase of this framework, and this can be enormously time consuming. Indeed, it has an overhead from 2 to 3 orders of magnitude on top of the application's execution time. For instance, the analysis can take up to hours or days in the case of large datasets used in real-world applications making this technique unfeasible.

To overcome the limitations mentioned above, we employed hardware-dependent characterization. We proposed NMPO, a high-level framework that employs machine learning models and hardware-dependent application characterization to predict the offloading suitability of unseen applications improving prediction times. We demonstrate how hardware-dependent analysis has an unnoticeable overhead over the application execution time. Compared to PISA it is much more practical and efficient. We then demonstrate the framework capability to decide application offloading suitability on a simulated near-memory computing system with an accuracy of 85.6%.

Limitations: even though the hardware-dependent characterization is much faster than its hardware-independent competitor, it results in less accuracy when employed in performance prediction models. Indeed, NMPO can predict offloading suitability as a first estimate, but it cannot accurately predict the resulting performance. Moreover, we also included critical radio-astronomical imaging kernels (Image-Domain Gridding) in this analysis showing how near-memory computing is not a viable option for obtaining performance gains.

This thesis contributes to the state-of-the-art showing that hardware-independent is a valuable methodology for characterizing applications. However, it is highly time-consuming. Indeed, we assess that hardware-dependent profiling tools are sufficiently good for providing application offloading insights with almost no additional overhead to the application runtime. Moreover, the number of available analysis tools is enormous. The path forward in the future would be to have a unified methodology and tool to profile workloads on different architectures easily.

7.1.2 Novel hardware architectures for radio-astronomical imaging

2D FFT is a critical kernel in real-world applications such as space and radio-astronomy. In particular, large 2D FFTs can be problematic in radio-astronomical imaging. Modern radio-telescopes such as LOFAR and SKA process extremely huges images that can be a critical bottleneck and sometimes can exceed the limited GPU's memory. Therefore, we profile large 2D FFTs employing hardware-dependent techniques showing how this kernel is memory bound. Then, we evaluate the performance of this critical workload on current CPU and GPU technology and near-memory computing prototypes on FPGAs. The NMC prototype on FPGA outperform CPUs by a factor of 120x; and we show similar performance on accelerators equipped with HBM, such as GPUs and the near-memory computing prototype.

Limitations: near-memory computing prototypes show interesting performance compared to high-end GPUs. However, developing the systems mentioned above is extremely time consuming, and the benefits do not justify the effort. High-Bandwidth Memory technology is a viable technology for memory-bound kernels such as FFT to obtain improved performance. Indeed, both the GPU and the NMC system reached noticeable performance improvements by exploiting this technology. Future GPU solutions would require larger memories to fit high-resolution images and high bandwidth for CPU-GPU and GPU-GPU communication.

Image-Domain Gridding is a time-critical section in state-of-the-art radio-astronomical imaging pipelines such as WSClean. As already mentioned, these kernels are compute bound and privilege high-performance accelerators such as GPUs and FPGAs. Additionally, radio-astronomical imaging needs high precision to individuate new objects in the sky, such as pulsars and stars. Reduced precision promises improved performance and unnoticeable results alterations compared to high-precision computation. We evaluate the feasibility of this technique for radio-astronomical imaging by evaluating the output quality in terms of SSIM (Structural Similarity Index Measure). We showed how reduce-precision could be employed to obtain high-quality images. Current state-of-the-art computing architectures such as CPUs and GPUs support a limited set of data types such as double-, single-, and half-precision floating point. On FPGAs, custom data types can be deployed to obtain further performance improvements. Therefore, we provide reduce-precision prototypes for radio-astronomical imaging on FPGAs using High-Level Synthesis, showing their improved performance against single-precision computing systems. More precisely, our reduced-precision implementation improves the throughput and energy efficiency of respectively 1.84x and 2.03x compared to the single-precision floating-point baseline on the same FPGA. Our solution is also 2.12x faster and 3.46x more energy-efficient than an Intel i9 9900k CPU and manages to keep up in throughput with an AMD RX 550 GPU.

Limitations: although the reduced-precision prototypes show improved performance compared to CPUs and single-precision FPGA prototypes, they struggle to keep GPUs' technology improvement pace. Nevertheless, this contribution shows how future hardware technologies could support radio-astronomical optimized data types to meet the Science Data Processor performance requirements.

This thesis shows that accelerators should be employed for radio-astronomical imaging. More precisely, GPUs should be a primary option because of their excellent performance for compute- and memory-bound applications. However, other architectures, such as future FPGAs with better floating-point and trigonometric function support, should be employed to prototype a custom solution. Moreover, reduced-precision results in an applicable methodology for a high-precision domain such as radio-astronomical imaging. The path forward for this research topic would be to evaluate these hardware technologies and methods in different parts of the radio-astronomical imaging pipeline to push forward the state-of-the-art.

7.2 HPC hardware trends

New memory technologies try to fill the gap created in the past years between processor and memory speed. Crucial solutions are represented by 2.5D and 3D stacked chip solutions. While the commonly called processing-in-memory (PIM) or near-data processing (NDP) produced niche products such as the Samsung PIM chip developed for AI workloads [229] and the UPMEM DRAM with processing units [230], important innovations will be integrated into more typical hardware using 2.5D solutions.
Apart from the AMD 3D-cache previously mentioned in *Chapter 1*, Intel plan to shortly release the first X86 CPU with HBM memory [231]. Note that CPUs with HBM are already available in the Fugaku SuperComputer [232]. The HBM bandwidth is far higher than the current DDR5 standard. This will definitely help CPUs process memory bounded application and regain popularity compared to GPUs.
The mentioned HBM-equipped CPU will also have the possibility to support the same time DDR5 memory adding a new layer of memory heterogeneity similar to what can already be found in FPGAs such as the Xilinx Alveo U280 [234].
High-performance CPUs were mainly CISC cores, which, compared to RISC, are highly complex. RISC cores were mainly employed in embedded systems to achieve low-power solutions [235, 236]. Nevertheless, the trend is now changing. Ampere started producing HPC CPU with ARM cores [237] and NVIDIA recently

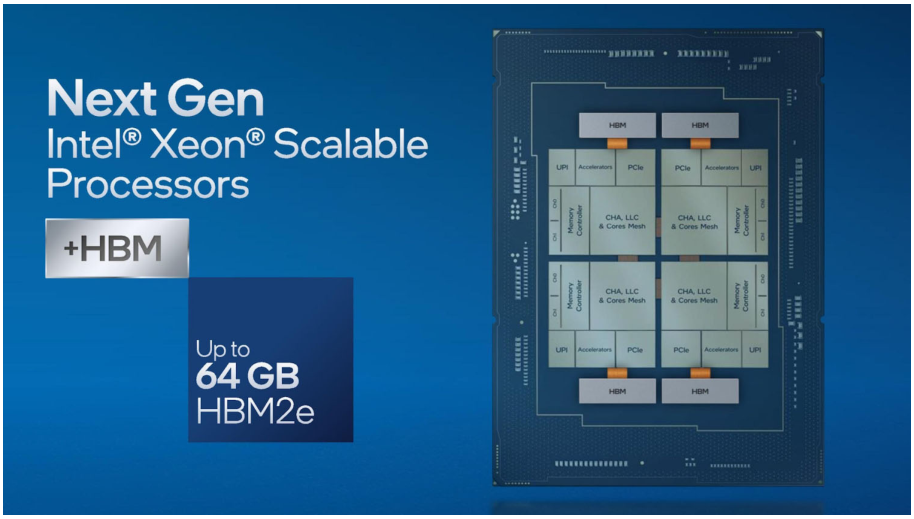


Figure 7.1: *Intel CPUs with HBM memory. The CPU consists of multiple chiplets (group of cores) with their own HBM memory. The CPU will support at the same time DDR5 memory [231].*

announced to enter the CPU market by introducing the next-generation Grace CPU and Hopper GPU. In particular, the NVIDIA solution can provide high-performance chips with a CPU and GPU connected through coherent memory up to 900 GB/s, which is enormously faster than PCIe5 [233]. They also propose dual CPUs with memory bandwidth up to 1 TB/s.

While a great effort is spent in bringing CPU to a similar performance level to accelerators, GPUs and FPGAs are improving too. GPUs become every year larger and equipped with faster memory, e.g., Hopper with 3 TB/s memory. However, a great limitation is the power consumption increase: upcoming GPUs will have a TDP of about 600-700 W compared to previous ones that were about 300-400 W. Additionally, they support special units to perform AI task with reduced precision and matrix operations efficiently. Today's FPGAs reached similar power consumption compared to GPUs. However, their performance in single-precision floating point are still far from GPUs. Indeed, new specific DSPs units are integrated into new generation FPGAs such as AMD/Xilinx Versal ACAP [12].

Artificial Intelligence and Machine Learning mainly drive these hardware trends. Indeed, these applications are employed in many research fields or even industries to develop digital twins. A digital twin is a digital copy of something that exists in reality and can be used to evaluate the past state of it and predict the future. This may be helpful in the field like weather forecasting, such as in the case of NVIDIA Omniverse [238] or also to predict the production level in a car factory.

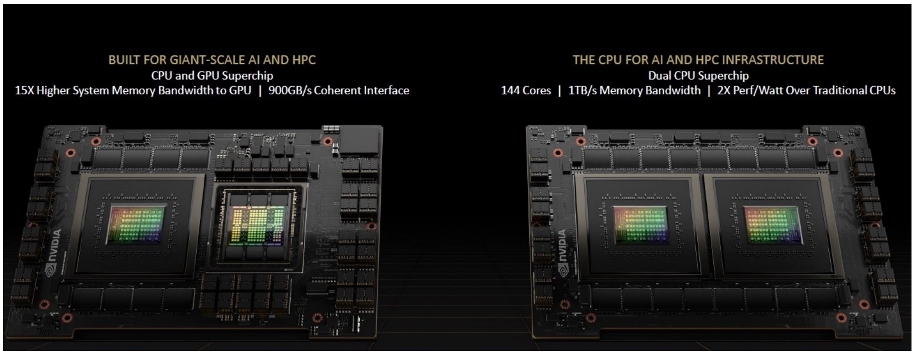

BUILT FOR GIANT-SCALE AI AND HPC
CPU and GPU Superchip
15X Higher System Memory Bandwidth to GPU | 900GB/s Coherent Interface
THE CPU FOR AI AND HPC INFRASTRUCTURE
Dual CPU Superchip
144 Cores | 1TB/s Memory Bandwidth | 2X Perf/Watt Over Traditional CPUs

Figure 7.2: *NVIDIA Grace-Hopper chips. On the left a Grace CPU with ARM cores and a Hopper GPUs connected with a coherent interface that can reach* 900 GB/s *and supporting high bandwidth memory. On the right 2 Grace CPUs with high bandwidth memory and fast interconnect [233].*

These upcoming hardware innovations will make the maintenance of existing software more challenging. Indeed, even if an automatic solution is desired, a large research effort must be spent on understanding how efficiently algorithms must adapt to new technologies in order not to waste computing power and energy. Some of these upcoming challenges are described in the next section.

7.3 Future work

This thesis work explored the varied landscape of available architectures for improving modern applications, focusing on radio-astronomical imaging. Nevertheless, there are still many open challenges that need to be addressed in the future.

Application characterization

Previous chapters show how hard and time consuming it is to perform hardware-independent characterization. An efficient and enough accurate solution is to employ hardware-dependent characterization. One main drawback of this technique is that each architecture is different, thus requiring, in worst cases, different tools.
Hardware vendors usually provides tools for profiling such as nvprof [62], CodeXL [63], Intel VTune [64]. However, it is hard to have a single tool or framework to analyze applications on multiple architectures. An example of similar attempts is Likwid [131], but it is limited to CPUs. Another weakness is the absence of standardized ways of characterizing applications. An example can be the roofline model [65], or more advanced models like X [239].

A challenge for the future would be to research and provide a common framework capable of profiling applications on various architectures with a standardized method. This would ease the application profiling tasks and provide a common ground for researchers interested in understanding application bottlenecks.

Approximate computing in HLS

We presented just one technique used in approximate computing in *Chapter 6*. Other approaches focus on building basic operations such as multiplication and division, which are common in today's workloads and time-consuming, exploiting solution approximation [240,241]. One of the main limitations of these approaches is that they are mainly provided as black boxes written in HDL languages. For instance, Ullah et al. [242, 243] proposed a novel approximate multiplier architecture designed specifically for FPGAs, which can be configured for different input sizes, achieving low power and reduced area results. Ebrahimi et al. [244] proposed an approximate multiplier and divider for fixed-point numbers exploiting the logarithmic approximation of these operations. Similar approaches have also been applied to floating point operations, such as Cheng et al. [245].

Similar to the templatized soft floating-point library in HLS [184], approximate operators can be expressed using High-Level Synthesis with low overhead but improving portability, readability, and usability. This would amplify the use of approximate computing in more application fields, bringing performance and power consumption benefits.

Optimization of other radio-astronomy applications

Image-Domain Gridding is almost perfectly optimized, reaching more than 85% of the peak performance of modern GPUs. This shift the bottleneck to other parts of the radio-astronomical imaging pipeline. For example, the CLEAN algorithm, as explained in *Chapter 2*, aims to remove the noise from the dirty sky image. This is done iteratively and can be highly time consuming based on how deep the cleaning is.

A preliminary analysis shows that CLEAN is memory bound (see the roofline model in *Figure 7.3*). Thus, CLEAN would be a great candidate for computation on accelerators such as FPGAs and GPUs. Furthermore, since Gridding/Degridding and FFTs are usually accelerated, CLEAN acceleration would also save communication time between the host and accelerators.

It should be mentioned that also other radio-astronomy kernels outside the imaging step can benefit from accelerated architectures, such as the calibration that tries to reduce the direction independent errors introduced by the instrument (radio-telescope) [103].

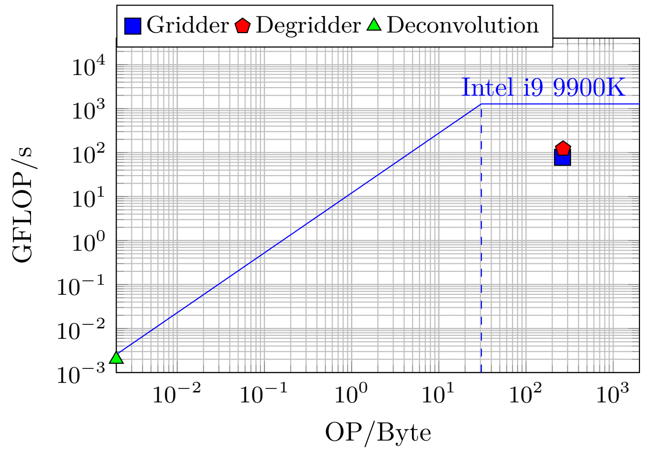


Figure 7.3: *Roofline Analysis of the main kernels in WSClean. As shown in the* Chapter 6 *the gridder and degridder kernel are compute bound. However, the deconvolution or CLEAN algorithm is mostly memory bound.*

AI Hardware for radio-astronomical imaging

GPUs have been equipped with hardware specifically optimized for Artificial Intelligence tasks in the past couple of years. One clear example is represented by tensor-cores units. Tensor cores are compute units that perform a matrix multiply and accumulate operation at warp level. These units exploit reduced-precision data-types representation to improve performance. The lastest generation of NVIDIA GPUs provides tensor-core units capable of performing up to 5-10x than in single precision. However, this computational model is typically applicable only to artificial intelligence workloads. A first attempt is represented by [246], where tensor-core units are employed for the first time for radio-astronomy applications. More precisely, Romein shows that considerable effort is needed to reshape the application into the tensor-core notation, but the gains are extremely interesting in speedup and energy consumption.
Therefore, since many radio-astronomical imaging algorithms such as gridding contains various multiply and accumulate operations, it makes sense to investigate the applicability of new AI hardware for the kernels mentioned above.

Machine Learning for radio-astronomical imaging

The CLEAN algorithms [109–111, 114, 247] are considered the state-of-the-art technique to remove instrument and observation noise from radio-astronomical images. Recently, a new approach called POLISH [248] has been presented. POLISH mixes super-resolution techniques and convolutional neural networks to obtain higher-resolution cleaned images. POLISH is currently able to work on synthetic data of the DSA-2000 radio telescope. We would like to investigate the feasibility of this technique on real datasets from LOFAR and/or SKA telescopes. Furthermore, we would like to compare its performance in terms of execution time and energy efficiency compared to WSClean.

Bibliography

[1] A. L'Heureux, K. Grolinger, H. F. Elyamany, and M. A. M. Capretz, "Machine learning with big data: Challenges and approaches," *IEEE Access*, vol. 5, pp. 7776–7797, 2017.

[2] T. Abdullah and A. Ahmet, "Genomics analyser: A big data framework for analysing genomics data," in *Proceedings of the Fourth IEEE/ACM International Conference on Big Data Computing, Applications and Technologies*, ser. BDCAT '17. New York, NY, USA: Association for Computing Machinery, 2017, p. 189–197. [Online]. Available: https://doi.org/10.1145/3148055.3148072

[3] A. Cavelan, R. M. Cabezón, M. Grabarczyk, and F. M. Ciorba, "A smoothed particle hydrodynamics mini-app for exascale," in *Proceedings of the Platform for Advanced Scientific Computing Conference*, ser. PASC '20. New York, NY, USA: Association for Computing Machinery, 2020. [Online]. Available: https://doi.org/10.1145/3394277.3401855

[4] B. Veenboer and J. Romein, "Radio-astronomical imaging on graphics processors," *Astronomy and Computing*, vol. 32, p. 100386, 2020. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S2213133720300408

[5] M. G. Labate, R. Braun, P. Dewdney, M. Waterson, and J. Wagg, "SKA1-LOW: Design and scientific objectives," in *2017 XXXIInd General Assembly and Scientific Symposium of the International Union of Radio Science (URSI GASS)*, 2017, pp. 1–4.

[6] R. Bolton, P. Broekema, T. Cornwell, G. van Diepen, C. Holli, M. Johnston-Holli *et al.*, "Parametric models of SDP compute requirements," *SKA Office, Manchester, UK, Tech. Rep. SKA-TEL-SDP-0000040*, vol. 21, 2016.

[7] W. A. Wulf and S. A. McKee, "Hitting the memory wall: Implications of the obvious," *SIGARCH Comput. Archit. News*, vol. 23, no. 1, p. 20–24, mar 1995. [Online]. Available: https://doi.org/10.1145/216585.216588

[8] J. L. Gustafson, *Moore's Law*. Boston, MA: Springer US, 2011, pp. 1177–1184. [Online]. Available: https://doi.org/10.1007/978-0-387-09766-4_81

[9] R. Dennard, F. Gaensslen, H.-N. Yu, V. Rideout, E. Bassous, and A. LeBlanc, "Design of ion-implanted mosfet's with very small physical dimensions," *IEEE Journal of Solid-State Circuits*, vol. 9, no. 5, pp. 256–268, 1974.

[10] D. Lee, K. Kim, K. Kim, H. Kim, J. Kim, Y. Park *et al.*, "25.2 a 1.2v 8gb 8-channel 128gb/s high-bandwidth memory (hbm) stacked dram with effective microbump i/o test methods using 29nm process and tsv," vol. 57, 02 2014, pp. 432–433.

[11] J. T. Pawlowski, "Hybrid memory cube (hmc)," in *2011 IEEE Hot Chips 23 Symposium (HCS)*, 2011, pp. 1–24.

[12] Xilinx. Xilinx Versal. [Online]. Available: https://www.xilinx.com/products/silicon-devices/acap/versal.html

[13] J. Choquette, W. Gandhi, O. Giroux, N. Stam, and R. Krashinsky, "NVIDIA A100 Tensor Core GPU: Performance and Innovation," *IEEE Micro*, vol. 41, no. 2, pp. 29–35, 2021.

[14] G. Singh, L. Chelini, S. Corda, A. J. Awan, S. Stuijk, R. Jordans *et al.*, "Near-memory computing: Past, present, and future," *Microprocessors and Microsystems*, vol. 71, p. 102868, 2019. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0141933119300389

[15] AMD. 3D-V Cache. [Online]. Available: https://www.amd.com/en/campaigns/3d-v-cache

[16] R. Swaminatha. Case Study: AMD products built with 3D packaging. [Online]. Available: https://hc33.hotchips.org/assets/program/tutorials/2021%20Hot%20Chips%20AMD%20Advanced%20Packaging%20Swaminathan%20Final%20%2020210820.pdf

[17] S. Daghaghi, N. Meisburger, M. Zhao, Y. Wu, S. Gobriel, C. Tai, and A. Shrivastava, "Accelerating SLIDE Deep Learning on Modern CPUs: Vectorization, Quantizations, Memory Optimizations, and More," 2021.

[18] A. Yazdanbakhsh, J. Park, H. Sharma, P. Lotfi-Kamran, and H. Esmaeilzadeh, "Neural acceleration for gpu throughput processors," in *2015 48th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2015, pp. 482–493.

[19] Intel. Intel Acquisition of Altera. [Online]. Available: https://newsroom.intel.com/press-kits/intel-acquisition-of-altera/

[20] AMD. AMD Completes Acquisition of Xilinx. [Online]. Available: https://www.amd.com/en/press-releases/2022-02-14-amd-completes-acquisition-xilinx

[21] G. Singh, J. Gómez-Luna, G. Mariani, G. F. Oliveira, S. Corda, S. Stuijk *et al.*, "NAPEL: Near-Memory Computing Application Performance Prediction via Ensemble Learning," in *2019 56th ACM/IEEE Design Automation Conference (DAC)*, 2019, pp. 1–6.

[22] Y. S. Shao and D. Brooks, "Isa-independent workload characterization and its implications for specialized architectures," in *2013 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, 2013, pp. 245–255.

[23] A. Anghel, L. M. Vasilescu, R. Jongerius, G. Dittmann, and G. Mariani, "An Instrumentation Approach for Hardware-Agnostic Software Characterization," in *Proceedings of the 12th ACM International Conference on Computing Frontiers*, ser. CF '15. New York, NY, USA: Association for Computing Machinery, 2015. [Online]. Available: https://doi.org/10.1145/2742854.2742859

[24] T. Grosser, H. Zheng, R. Aloor, A. Simbürger, A. Größlinger, and L.-N. Pouchet, "Polly-polyhedral optimization in llvm," in *Proceedings of the First International Workshop on Polyhedral Compilation Techniques (IMPACT)*, vol. 2011, 2011, p. 1.

[25] J. Dongarra and P. Luszczek, *TOP500*. Boston, MA: Springer US, 2011, pp. 2055–2057. [Online]. Available: https://doi.org/10.1007/978-0-387-09766-4_157

[26] J. Dongarra, M. Heroux, and P. Luszczek, "High-performance conjugate-gradient benchmark: A new metric for ranking high-performance computing systems," *International Journal of High Performance Computing Applications*, vol. 30, 08 2015.

[27] O. Sarood, A. Langer, A. Gupta, and L. Kale, "Maximizing throughput of overprovisioned hpc data centers under a strict power budget," in *SC '14: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, 2014, pp. 807–818.

[28] S. Cherubin and G. Agosta, "Tools for Reduced Precision Computation: A Survey," *ACM Comput. Surv.*, vol. 53, no. 2, Apr. 2020. [Online]. Available: https://doi.org/10.1145/3381039

[29] L. Chelini, O. Zinenko, T. Grosser, and H. Corporaal, "Declarative loop tactics for domain-specific optimization," *ACM Trans. Archit. Code Optim.*, vol. 16, no. 4, dec 2019. [Online]. Available: https://doi.org/10.1145/3372266

[30] S. Corda, G. Singh, A. J. Awan, R. Jordans, and H. Corporaal, "Memory and Parallelism Analysis Using a Platform-Independent Approach," in *Proceedings of the 22nd International Workshop on Software and Compilers for Embedded Systems*, ser. SCOPES '19. New York, NY, USA: Association for Computing Machinery, 2019, p. 23–26. [Online]. Available: https://doi.org/10.1145/3323439.3323988

[31] ——, "Platform Independent Software Analysis for Near Memory Computing," in *2019 22nd Euromicro Conference on Digital System Design (DSD)*, 2019, pp. 606–609.

[32] S. Corda, M. Kumaraswamy, A. J. Awan, R. Jordans, A. Kumar, and H. Corporaal, "Nmpo: Near-memory computing profiling and offloading," in *2021 24th Euromicro Conference on Digital System Design (DSD)*, 2021, pp. 259–267.

[33] S. Corda, B. Veenboer, A. J. Awan, A. Kumar, R. Jordans, and H. Corporaal, "Near memory acceleration on high resolution radio astronomy imaging," in *2020 9th Mediterranean Conference on Embedded Computing (MECO)*, 2020, pp. 1–6.

[34] S. Corda, B. Veenboer, A. J. Awan, J. W. Romein, R. Jordans, A. Kumar *et al.*, "Reduced-precision acceleration of radio-astronomical imaging on reconfigurable hardware," *IEEE Access*, vol. 10, pp. 22 819–22 843, 2022.

[35] K. Rupp, "Microprocessor Trend Data," https://github.com/karlrupp/microprocessor-trend-data.

[36] C. A. Mack, "Fifty years of moore's law," *IEEE Transactions on Semiconductor Manufacturing*, vol. 24, no. 2, pp. 202–207, 2011.

[37] S. Daghaghi, N. Meisburger, M. Zhao, Y. Wu, S. Gobriel, C. Tai, and A. Shrivastava, "Accelerating slide deep learning on modern cpus: Vectorization, quantizations, memory optimizations, and more," 2021. [Online]. Available: https://arxiv.org/abs/2103.10891

[38] Apple. Apple unveils m2, taking the breakthrough performance and capabilities of m1 even further. https://www.apple.com/newsroom/2022/06/apple-unveils-m2-with-breakthrough-performance-and-capabilities/, 2022.

[39] AMD. AMD "Zen" Core Architecture. [Online]. Available: https://www.amd.com/en/technologies/zen-core

[40] J. L. Hennessy and D. A. Patterson, *Computer Architecture: A Quantitative Approach*, 5th ed. Amsterdam: Morgan Kaufmann, 2012.

[41] J. E. Stone, D. Gohara, and G. Shi, "Opencl: A parallel programming standard for heterogeneous computing systems," *Computing in Science Engineering*, vol. 12, no. 3, pp. 66–73, 2010.

[42] NVIDIA. Cuda c++ programming guide. https://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html, 2022.

[43] Z. Jia, M. Maggioni, B. Staiger, and D. P. Scarpazza, "Dissecting the nvidia volta gpu architecture via microbenchmarking," 2018. [Online]. Available: https://arxiv.org/abs/1804.06826

[44] A. Li, S. L. Song, M. Wijtvliet, A. Kumar, and H. Corporaal, "SFU-Driven Transparent Approximation Acceleration on GPUs," in *Proceedings of the 2016 International Conference on Supercomputing*, ser. ICS '16. New York, NY, USA: Association for Computing Machinery, 2016. [Online]. Available: https://doi.org/10.1145/2925426.2926255

[45] P. Kharya, "TensorFloat-32 in the A100 GPU Accelerates AI Training, HPC up to 20x," https://blogs.nvidia.com/blog/2020/05/14/tensorfloat-32-precision-format/, 2020.

[46] H. S. Stone, "A logic-in-memory computer," *IEEE Transactions on Computers*, vol. C-19, no. 1, pp. 73–78, 1970.

[47] D. Elliott, W. Snelgrove, and M. Stumm, "Computational ram: A memory-simd hybrid and its application to dsp," in *1992 Proceedings of the IEEE Custom Integrated Circuits Conference*, 1992, pp. 30.6.1–30.6.4.

[48] P. M. Kogge, "Execube-a new architecture for scaleable mpps," in *1994 International Conference on Parallel Processing Vol. 1*, vol. 1, 1994, pp. 77–84.

[49] M. Gokhale, B. Holmes, and K. Iobst, "Processing in memory: the terasys massively parallel pim array," *Computer*, vol. 28, no. 4, pp. 23–31, 1995.

[50] C. Kozyrakis, S. Perissakis, D. Patterson, T. Anderson, K. Asanovic, N. Cardwell *et al.*, "Scalable processors in the billion-transistor era: Iram," *Computer*, vol. 30, no. 9, pp. 75–78, 1997.

[51] R. Nair, S. F. Antao, C. Bertolli, P. Bose, J. R. Brunheroto, T. Chen *et al.*, "Active memory cube: A processing-in-memory architecture for exascale systems," *IBM Journal of Research and Development*, vol. 59, no. 2/3, pp. 17:1–17:14, 2015.

[52] J. Jeddeloh and B. Keeth, “Hybrid memory cube new dram architecture increases density and performance,” in *2012 Symposium on VLSI Technology (VLSIT)*, 2012, pp. 87–88.

[53] A. Boroumand, S. Ghose, Y. Kim, R. Ausavarungnirun, E. Shiu, R. Thakur *et al.*, “Google Workloads for Consumer Devices: Mitigating Data Movement Bottlenecks,” *SIGPLAN Not.*, vol. 53, no. 2, pp. 316–331, Mar. 2018. [Online]. Available: http://doi.acm.org/10.1145/3296957.3173177

[54] M. Torabzadehkashi, S. Rezaei, V. Alves, and N. Bagherzadeh, “Compstor: An in-storage computation platform for scalable distributed processing,” in *2018 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, 2018, pp. 1260–1267.

[55] J. Liu, H. Zhao, M. A. Ogleari, D. Li, and J. Zhao, “Processing-in-memory for energy-efficient neural network training: A heterogeneous approach,” in *2018 51st Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2018, pp. 655–668.

[56] CMU-SAFARI, “ZSim+Ramulator - A Processing-in-Memory Simulation Framework,” https://github.com/CMU-SAFARI/ramulator-pim, 2020.

[57] L. Ke, U. Gupta, B. Y. Cho, D. Brooks, V. Chandra, U. Diril *et al.*, “RecNMP:Accelerating Personalized Recommendation with Near-Memory Processing,” in *2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA)*, 2020, pp. 790–803.

[58] B. Y. Cho, Y. Kwon, S. Lym, and M. Erez, “Near data acceleration with concurrent host access,” in *2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA)*, 2020, pp. 818–831.

[59] C. T. Apple, “Evaluation and performance of computers: The program monitor&mdash;a device for program performance measurement,” in *Proceedings of the 1965 20th National Conference*, ser. ACM ’65. New York, NY, USA: ACM, 1965, pp. 66–75. [Online]. Available: http://doi.acm.org/10.1145/800197.806034

[60] A. Srivastava and A. Eustace, “Atom: A system for building customized program analysis tools,” in *Proceedings of the ACM SIGPLAN 1994 Conference on Programming Language Design and Implementation*, ser. PLDI ’94. New York, NY, USA: Association for Computing Machinery, 1994, p. 196–205. [Online]. Available: https://doi.org/10.1145/178243.178260

[61] K. Ali and O. Lhoták, “Application-only call graph construction,” vol. 7313, 06 2012, pp. 688–712.

[62] NVIDIA, “CUDA toolkit documentation - nvprof,” https://docs.nvidia.com/cuda/profiler-users-guide/index.html.

[63] AMD, “AMD CodeXL,” https://github.com/GPUOpen-Archive/CodeXL.

[64] INTEL. Intel VTune Amplifier. [Online]. Available: https://software.intel.com/en-us/intel-vtune-amplifier-xe

[65] S. Williams, A. Waterman, and D. Patterson, “Roofline: an insightful visual performance model for multicore architectures,” *Communications of the ACM*, vol. 52, no. 4, pp. 65–76, 2009.

[66] Y. J. Lo, S. Williams, B. V. Straalen, T. J. Ligocki, M. J. Cordery, N. J. Wright *et al.*, “Roofline model toolkit: A practical tool for architectural and program analysis,” in *International Workshop on Performance Modeling, Benchmarking and Simulation of High Performance Computer Systems.* Springer, 2014, pp. 129–148.

[67] G. Mariani, A. Anghel, R. Jongerius, and G. Dittmann, “Scaling application properties to exascale,” in *Proceedings of the 12th ACM International Conference on Computing Frontiers*, ser. CF ’15. New York, NY, USA: Association for Computing Machinery, 2015. [Online]. Available: https://doi.org/10.1145/2742854.2742860

[68] C. Lattner and V. Adve, “LLVM: a compilation framework for lifelong program analysis amp; transformation,” in *International Symposium on Code Generation and Optimization, 2004. CGO 2004.*, 2004, pp. 75–86.

[69] U. Bora, S. Vaishay, S. Joshi, and R. Upadrasta, “Openmp aware mhp analysis for improved static data-race detection,” in *2021 IEEE/ACM 7th Workshop on the LLVM Compiler Infrastructure in HPC (LLVM-HPC).* IEEE, 2021, pp. 1–11.

[70] T. Grosser, A. Groesslinger, and C. Lengauer, “Polly—performing polyhedral optimizations on a low-level intermediate representation,” *Parallel Processing Letters*, vol. 22, no. 04, p. 1250010, 2012.

[71] D. Zhang, N. Jayasena, A. Lyashevsky, J. L. Greathouse, L. Xu, and M. Ignatowski, “TOP-PIM: Throughput-Oriented Programmable Processing in Memory,” in *Proceedings of the 23rd International Symposium on High-Performance Parallel and Distributed Computing*, ser. HPDC ’14. New York, NY, USA: Association for Computing Machinery, 2014, p. 85–98. [Online]. Available: https://doi.org/10.1145/2600212.2600213

[72] Y. Yu, W. Wang, J. Zhang, and K. Ben Letaief, “LRC: Dependency-aware cache management for data analytics clusters,” *Proceedings - IEEE INFOCOM*, 2017.

[73] B. Gregg, “Performance counters,” *2018 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, 2018. [Online]. Available: http://www.brendangregg.com/perf.html

[74] D. Terpstra, H. Jagode, H. You, and J. Dongarra, “Collecting performance data with papi-c,” *Proceedings of the 3rd International Workshop on Parallel Tools for High Performance Computing 2009*, pp. 157–173, 05 2010.

[75] Y. Wada, Y. He, T. Cao, and M. Kondo, “A power management framework with simple dsl for automatic power-performance optimization on power-constrained hpc systems,” 03 2018, pp. 199–218.

[76] V. J. Reddi, A. Settle, D. A. Connors, and R. S. Cohn, “Pin: A binary instrumentation tool for computer architecture research and education,” in *Proceedings of the 2004 Workshop on Computer Architecture Education: Held in Conjunction with the 31st International Symposium on Computer Architecture*, ser. WCAE ’04. New York, NY, USA: Association for Computing Machinery, 2004, p. 22–es. [Online]. Available: https://doi.org/10.1145/1275571.1275600

[77] Y. Kim, W. Yang, and O. Mutlu, “Ramulator: A Fast and Extensible DRAM Simulator,” *IEEE Computer Architecture Letters*, vol. 15, no. 1, pp. 45–49, Jan 2016.

[78] S. Li, Z. Yang, D. Reddy, A. Srivastava, and B. Jacob, “Dramsim3: A cycle-accurate, thermal-capable dram simulator,” *IEEE Computer Architecture Letters*, vol. 19, no. 2, pp. 106–109, 2020.

[79] N. Binkert, B. Beckmann, G. Black, S. K. Reinhardt, A. Saidi, A. Basu *et al.*, “The Gem5 Simulator,” *SIGARCH Comput. Archit. News*, vol. 39, no. 2, p. 1–7, Aug. 2011. [Online]. Available: https://doi.org/10.1145/2024716.2024718

[80] D. Sanchez and C. Kozyrakis, “ZSim: Fast and Accurate Microarchitectural Simulation of Thousand-Core Systems,” in *Proceedings of the 40th Annual International Symposium on Computer Architecture*, ser. ISCA ’13. New York, NY, USA: Association for Computing Machinery, 2013, p. 475–486. [Online]. Available: https://doi.org/10.1145/2485922.2485963

[81] M. Khairy, Z. Shen, T. M. Aamodt, and T. G. Rogers, “Accel-sim: An extensible simulation framework for validated gpu modeling,” in *2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA)*, 2020, pp. 473–486.

[82] T. Gysi, T. Grosser, L. Brandner, and T. Hoefler, “A fast analytical model of fully associative caches,” in *Proceedings of the 40th ACM SIGPLAN Conference on Programming Language Design and Implementation*, ser. PLDI 2019. New York, NY, USA: Association for Computing Machinery, 2019, p. 816–829. [Online]. Available: https://doi.org/10.1145/3314221.3314606

[83] R. Jordans, R. Corvino, L. Jóźwiak, and H. Corporaal, “Exploring processor parallelism: Estimation methods and optimization strategies,” in *2013 IEEE 16th International Symposium on Design and Diagnostics of Electronic Circuits Systems (DDECS)*, 2013, pp. 18–23.

[84] J. F. Eusse, L. G. Murillo, C. McGirr, R. Leupers, and G. Ascheid, “Application-specific architecture exploration based on processor-agnostic performance estimation,” in *Proceedings of the 18th International Workshop on Software and Compilers for Embedded Systems*, ser. SCOPES ’15. New York, NY, USA: Association for Computing Machinery, 2015, p. 84–87. [Online]. Available: https://doi.org/10.1145/2764967.2771932

[85] V. Cabezas, “A tool for analysis and visualization of application properties,” Technical Report RZ3834, IBM, Tech. Rep., 2012.

[86] Wikipedia. Vector Fabrics. [Online]. Available: https://en.wikipedia.org/wiki/Vector_Fabrics,_B.V.

[87] G. S. Tjaden and M. J. Flynn, “Detection and parallel execution of independent instructions,” *IEEE Transactions on Computers*, vol. C-19, pp. 889–895, 1970.

[88] K. B. Theobald, G. R. Gao, and L. J. Hendren, “On the limits of program parallelism and its smoothability,” *SIGMICRO Newsl.*, vol. 23, no. 1-2, pp. 10–19, Dec. 1992. [Online]. Available: http://doi.acm.org/10.1145/144965.144977

[89] S. L. Graham, P. B. Kessler, and M. K. Mckusick, “Gprof: A call graph execution profiler,” *SIGPLAN Not.*, vol. 17, no. 6, pp. 120–126, Jun. 1982. [Online]. Available: http://doi.acm.org/10.1145/872726.806987

[90] K. Hoste and L. Eeckhout, “Microarchitecture-independent workload characterization,” *IEEE Micro*, vol. 27, no. 3, pp. 63–72, May 2007. [Online]. Available: http://dx.doi.org/10.1109/MM.2007.56

[91] ——, “Comparing Benchmarks Using Key Microarchitecture-Independent Characteristics,” *2006 IEEE International Symposium on Workload Characterization*, pp. 83–92, 2006.

[92] C.-K. Luk, R. Cohn, R. Muth, H. Patil, A. Klauser, G. Lowney *et al.*, “Pin: Building customized program analysis tools with dynamic instrumentation,” in *Proceedings of the 2005 ACM SIGPLAN Conference on Programming Language Design and Implementation*, ser. PLDI ’05. New York, NY, USA: ACM, 2005, pp. 190–200. [Online]. Available: http://doi.acm.org/10.1145/1065010.1065034

[93] V. Caparrós Cabezas and P. Stanley-Marbell, “Parallelism and data movement characterization of contemporary application classes,” in *Proceedings of the Twenty-third Annual ACM Symposium on Parallelism in Algorithms and Architectures*, ser. SPAA ’11. New York, NY, USA: ACM, 2011, pp. 95–104. [Online]. Available: http://doi.acm.org/10.1145/1989493.1989506

[94] M. Ferdman, A. Adileh, O. Kocberber, S. Volos, M. Alisafaee, D. Jevdjic *et al.*, “Clearing the clouds: A study of emerging scale-out workloads on modern hardware,” *SIGPLAN Not.*, vol. 47, no. 4, pp. 37–48, Mar. 2012. [Online]. Available: http://doi.acm.org/10.1145/2248487.2150982

[95] IBM. (2012) Evaluate performance for linux on power. [Online]. Available: https://www.ibm.com/developerworks/linux/library/l-evaluatelinuxonpower/

[96] A. Yasin, “A top-down method for performance analysis and counters architecture,” in *2014 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, 2014, pp. 35–44.

[97] INTEL. Intel advisor roofline. [Online]. Available: https://software.intel.com/en-us/articles/intel-advisor-roofline

[98] van Haarlem, M. P., Wise, M. W., Gunst, A. W., Heald, G., McKean, J. P., Hessels, J. W. T. *et al.*, “LOFAR: The LOw-Frequency ARray,” *A&A*, vol. 556, p. A2, 2013. [Online]. Available: https://doi.org/10.1051/0004-6361/201220873

[99] R. J. Selina, E. J. Murphy, M. McKinnon, A. Beasley, B. Butler, C. Carilli *et al.*, “The Next Generation Very Large Array: A Technical Overview,” 2018.

[100] J. L. Jonas, “MeerKAT,” in *2019 URSI Asia-Pacific Radio Science Conference (AP-RASC)*, 2019, pp. 1–1.

[101] S. W. Schediwy, D. R. Gozzard, S. Stobie, C. Gravestock, T. Pulbrook, R. Whitaker *et al.*, "Phase Synchronization for the Mid-Frequency Square Kilometre Array Telescope," in *2018 2nd URSI Atlantic Radio Science Meeting (AT-RASC)*, 2018, pp. 1–1.

[102] Smirnov, O. M., "Revisiting the radio interferometer measurement equation - i. a full-sky jones formalism," *A&A*, vol. 527, p. A106, 2011. [Online]. Available: https://doi.org/10.1051/0004-6361/201016082

[103] U. Rau, S. Bhatnagar, M. A. Voronkov, and T. J. Cornwell, "Advances in Calibration and Imaging Techniques in Radio Interferometry," *Proceedings of the IEEE*, vol. 97, no. 8, pp. 1472–1481, 2009.

[104] B. Veenboer and J. W. Romein, "Radio-astronomical imaging: Fpgas vs gpus," in *Euro-Par 2019: Parallel Processing*, R. Yahyapour, Ed. Cham: Springer International Publishing, 2019, pp. 509–521.

[105] S. van der Tol, B. Veenboer, and A. R. Offringa, "Image Domain Gridding: a fast method for convolutional resampling of visibilities," *Astronomy & Astrophysics*, vol. 616, p. A27, Aug 2018. [Online]. Available: http://dx.doi.org/10.1051/0004-6361/201832858

[106] A. R. Offringa, F. Mertens, S. van der Tol, B. Veenboer, B. K. Gehlot, L. V. E. Koopmans, and M. Mevius, "Precision requirements for interferometric gridding in the analysis of a 21 cm power spectrum," *Astronomy & Astrophysics*, vol. 631, p. A12, Oct 2019. [Online]. Available: http://dx.doi.org/10.1051/0004-6361/201935722

[107] T. J. Cornwell, K. Golap, and S. Bhatnagar, "The Noncoplanar Baselines Effect in Radio Interferometry: The W-Projection Algorithm," *IEEE Journal of Selected Topics in Signal Processing*, vol. 2, no. 5, pp. 647–657, 2008.

[108] S. Bhatnagar, T. J. Cornwell, K. Golap, and J. M. Uson, "Correcting direction-dependent gains in the deconvolution of radio interferometric images," *Astronomy & Astrophysics*, vol. 487, no. 1, p. 419–429, Jun 2008. [Online]. Available: http://dx.doi.org/10.1051/0004-6361:20079284

[109] B. G. Clark, "An efficient implementation of the algorithm 'CLEAN'," *Astronomy & Astrophysics*, vol. 89, no. 3, p. 377, Sep. 1980.

[110] F. Schwab, "Relaxing the isoplanatism assumption in self-calibration; applications to low-frequency radio interferometry," *The Astronomical Journal*, vol. 89, pp. 1076–1081, 1984.

[111] T. J. Cornwell, "Multiscale CLEAN Deconvolution of Radio Synthesis Images," *IEEE Journal of Selected Topics in Signal Processing*, vol. 2, no. 5, p. 793–801, Oct 2008. [Online]. Available: http://dx.doi.org/10.1109/JSTSP.2008.2006388

[112] A. R. Offringa and O. Smirnov, "An optimized algorithm for multiscale wideband deconvolution of radio astronomical images," *Monthly Notices of the Royal Astronomical Society*, vol. 471, no. 1, p. 301–316, Jun 2017. [Online]. Available: http://dx.doi.org/10.1093/mnras/stx1547

[113] J. W. Rich, W. J. G. de Blok, T. J. Cornwell, E. Brinks, F. Walter, I. Bagetakos, and R. C. Kennicutt, "Multi-Scale CLEAN: A comparison of its performance against classical CLEAN in galaxies using THINGS," *The Astronomical Journal*, vol. 136, no. 6, p. 2897–2920, Nov 2008. [Online]. Available: http://dx.doi.org/10.1088/0004-6256/136/6/2897

[114] A. R. Offringa, B. McKinley, N. Hurley-Walker, F. H. Briggs, R. B. Wayth, D. L. Kaplan *et al.*, "WSClean: an implementation of a fast, generic wide-field imager for radio astronomy," *Monthly Notices of the Royal Astronomical Society*, vol. 444, no. 1, p. 606–619, Aug 2014. [Online]. Available: http://dx.doi.org/10.1093/mnras/stu1368

[115] C. E. Shannon, "Prediction and entropy of printed English," *The Bell System Technical Journal*, vol. 30, no. 1, pp. 50–64, Jan 1951.

[116] L. Yen, S. C. Draper, and M. D. Hill, "Notary: Hardware Techniques to Enhance Signatures," in *roceedings of the 41st Annual IEEE/ACM International Symposium on Microarchitecture*, ser. MICRO 41. Washington, DC, USA: IEEE Computer Society, 2008, pp. 234–245. [Online]. Available: https://doi.org/10.1109/MICRO.2008.4771794

[117] X. Gu, I. Christopher, T. Bai, C. Zhang, and C. Ding, “A Component Model of Spatial Locality,” in *Proceedings of the 2009 International Symposium on Memory Management*, ser. ISMM ’09. New York, NY, USA: ACM, 2009, pp. 99–108. [Online]. Available: http://doi.acm.org/10.1145/1542431.1542446

[118] I. Jolliffe, *Principal Component Analysis*. Springer Verlag, 1986.

[119] IBM. IBM Power 9. [Online]. Available: https://www.ibm.com/it-infrastructure/power/power9

[120] J. Ahn *et al.*, “A scalable processing-in-memory accelerator for parallel graph processing,” in *2015 ACM/IEEE 42nd Annual International Symposium on Computer Architecture (ISCA)*, June 2015, pp. 105–117.

[121] M. Gao, G. Ayers, and C. Kozyrakis, “Practical Near-Data Processing for In-Memory Analytics Frameworks,” in *2015 International Conference on Parallel Architecture and Compilation (PACT)*, 2015, pp. 113–124.

[122] A. Anghel, L. M. Vasilescu, G. Mariani, R. Jongerius, and G. Dittmann, “An instrumentation approach for hardware-agnostic software characterization,” *International Journal of Parallel Programming*, vol. 44, no. 5, pp. 924–948, Oct 2016.

[123] Cavazos-lab, “PolyBench-ACC,” https://github.com/cavazos-lab/PolyBench-ACC, 2016.

[124] S. Che, M. Boyer, J. Meng, D. Tarjan, J. W. Sheaffer, S. Lee, and K. Skadron, “Rodinia: A benchmark suite for heterogeneous computing,” in *2009 IEEE International Symposium on Workload Characterization (IISWC)*, Oct 2009, pp. 44–54.

[125] K. Asanovic, R. Bodik, B. C. Catanzaro, J. J. Gebis, P. Husbands, K. Keutzer *et al.*, “The landscape of parallel computing research: A view from berkeley,” Technical Report UCB/EECS-2006-183, EECS Department, University of . . . , Tech. Rep., 2006.

[126] A. J. Awan, M. Ohara, E. Ayguade, K. Ishizaki, M. Brorsson, and V. Vlassov, “Identifying the Potential of near Data Processing for Apache Spark,” in *Proceedings of the International Symposium on Memory Systems*, ser. MEMSYS ’17. New York, NY, USA: Association for Computing Machinery, 2017, p. 60–67. [Online]. Available: https://doi.org/10.1145/3132402.3132427

[127] J. Ahn, S. Yoo, O. Mutlu, and K. Choi, “PIM-enabled instructions: a low-overhead, locality-aware processing-in-memory architecture,” in *Proceedings of the 42nd Annual International Symposium on Computer Architecture*. ACM, 2015, pp. 336–348.

[128] A. J. Awan, “Performance Characterization and Optimization of In-Memory Data Analytics on a Scale-up Server,” Ph.D. dissertation, KTH Royal Institute of Technology and Universitat Politècnica de Catalunya, 2017.

[129] A. Pattnaik, X. Tang, A. Jog, O. Kayiran, A. K. Mishra, M. T. Kandemir *et al.*, “Scheduling techniques for GPU architectures with processing-in-memory capabilities,” in *2016 International Conference on Parallel Architecture and Compilation Techniques (PACT)*, 2016, pp. 31–44.

[130] K. Hsieh, E. Ebrahim, G. Kim, N. Chatterjee, M. O’Connor, N. Vijaykumar *et al.*, “Transparent Offloading and Mapping (TOM): Enabling Programmer-Transparent Near-Data Processing in GPU Systems,” in *2016 ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA)*, 2016.

[131] J. Treibig, G. Hager, and G. Wellein, “LIKWID: A Lightweight Performance-Oriented Tool Suite for X86 Multicore Environments,” in *Proceedings of the 2010 39th International Conference on Parallel Processing Workshops*, ser. ICPPW ’10. USA: IEEE Computer Society, 2010, p. 207–216. [Online]. Available: https://doi.org/10.1109/ICPPW.2010.38

[132] P. Gu, X. Xie, Y. Ding, G. Chen, W. Zhang, D. Niu, and Y. Xie, “iPIM: Programmable In-Memory Image Processing Accelerator Using Near-Bank Architecture,” in *2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA)*, 2020, pp. 804–817.

[133] B. Y. Cho, Y. Kwon, S. Lym, and M. Erez, “Near Data Acceleration with Concurrent Host Access,” in *Proceedings of the ACM/IEEE 47th Annual International Symposium*

on Computer Architecture, ser. ISCA '20. IEEE Press, 2020, p. 818–831. [Online]. Available: https://doi.org/10.1109/ISCA45697.2020.00072

[134] Y. L. Karthik Chandrasekar, Christian Weis and K. Goossens, "DRAMPower: Open-source DRAM Power & Energy Estimation Tool." [Online]. Available: http://www.drampower.info

[135] T. G. Dietterich, "Ensemble Methods in Machine Learning," in *Proceedings of the First International Workshop on Multiple Classifier Systems*, ser. MCS '00. Berlin, Heidelberg: Springer-Verlag, 2000, p. 1–15.

[136] L. Breiman, "Random Forests," *Mach. Learn.*, vol. 45, no. 1, p. 5–32, Oct. 2001. [Online]. Available: https://doi.org/10.1023/A:1010933404324

[137] A. Géron, *Hands-on machine learning with Scikit-Learn and TensorFlow : concepts, tools, and techniques to build intelligent systems.* Sebastopol, CA: O'Reilly Media, 2017.

[138] C. Weis, A. Mutaal, O. Naji, M. Jung, A. Hansson, and N. Wehn, "DRAMSpec: A High-Level DRAM Timing, Power and Area Exploration Tool," *Int. J. Parallel Program.*, vol. 45, no. 6, p. 1566–1591, Dec. 2017. [Online]. Available: https://doi.org/10.1007/s10766-016-0473-y

[139] S. Grauer-Gray, L. Xu, R. Searles, S. Ayalasomayajula, and J. Cavazos, "Auto-tuning a high-level language targeted to GPU codes," in *2012 Innovative Parallel Computing (InPar)*, 2012, pp. 1–10.

[140] ASTRON, "Image Domain Gridding (IDG)," https://gitlab.com/astron-idg/idg, 2020.

[141] G. Ofenbeck, R. Steinmann, V. Caparros, D. G. Spampinato, and M. Püschel, "Applying the roofline model," in *2014 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS).* IEEE, 2014, pp. 76–85.

[142] W. Kirch, Ed., *Pearson's Correlation Coefficient.* Dordrecht: Springer Netherlands, 2008, pp. 1090–1091. [Online]. Available: https://doi.org/10.1007/978-1-4020-5614-7_2569

[143] J. Ahn, S. Yoo, O. Mutlu, and K. Choi, "PIM-enabled instructions: A low-overhead, locality-aware processing-in-memory architecture," in *2015 ACM/IEEE 42nd Annual International Symposium on Computer Architecture (ISCA)*, 2015.

[144] R. Hadidi, L. Nai, H. Kim, and H. Kim, "CAIRO: A Compiler-Assisted Technique for Enabling Instruction-Level Offloading of Processing-In-Memory," *ACM Trans. Archit. Code Optim.*, 2017.

[145] H. Ahmed, P. C. Santos, J. P. C. Lima, R. F. Moura, M. A. Z. Alves, A. C. S. Beck, and L. Carro, "A Compiler for Automatic Selection of Suitable Processing-in-Memory Instructions," in *2019 Design, Automation Test in Europe Conference Exhibition (DATE)*, 2019, pp. 564–569.

[146] L. Nai, R. Hadidi, J. Sim, H. Kim, P. Kumar, and H. Kim, "GraphPIM: Enabling Instruction-Level PIM Offloading in Graph Computing Frameworks," in *2017 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, 2017, pp. 457–468.

[147] P. J. Joseph, Kapil Vaswani, and M. J. Thazhuthaveetil, "Construction and use of linear regression models for processor performance analysis," in *The Twelfth International Symposium on High-Performance Computer Architecture, 2006.*, 2006, pp. 99–108.

[148] A. Calotoiu, T. Hoefler, M. Poke, and F. Wolf, "Using automated performance modeling to find scalability bugs in complex codes," in *SC '13: Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, 2013, pp. 1–12.

[149] P. E. Bailey, D. K. Lowenthal, V. Ravi, B. Rountree, M. Schulz, and B. R. De Supinski, "Adaptive Configuration Selection for Power-Constrained Heterogeneous Systems," in *2014 43rd International Conference on Parallel Processing*, 2014, pp. 371–380.

[150] G. Wu, J. L. Greathouse, A. Lyashevsky, N. Jayasena, and D. Chiou, "GPGPU performance and power estimation using machine learning," in *2015 IEEE 21st International Symposium on High Performance Computer Architecture (HPCA)*, 2015, pp. 564–576.

[151] G. Mariani, A. Anghel, R. Jongerius, and G. Dittmann, "Predicting Cloud Performance for HPC Applications: A User-Oriented Approach," in *2017 17th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID)*, 2017, pp. 524–533.

[152] R. F. Araujo *et al.*, "A CPI breakdown model plug-in for optimizing application performance," in *2013 3rd International Workshop on Developing Tools as Plug-Ins (TOPI)*, May 2013, pp. 31–36.

[153] J. v. Lunteren *et al.*, "Coherently Attached Programmable Near-Memory Acceleration Platform and its application to Stencil Processing," in *2019 Design, Automation Test in Europe Conference Exhibition (DATE)*, March 2019, pp. 668–673.

[154] ——, "Designing a Programmable Wire-Speed Regular-Expression Matching Accelerator," in *2012 45th Annual IEEE/ACM International Symposium on Microarchitecture*, Dec 2012, pp. 461–472.

[155] OpenPower, "Power ISA 3.0," https://openpowerfoundation.org/?resource_lib=power-isa-version-3-0.

[156] M. Parker, "Understanding Peak Floating-Point Performance Claims." [Online]. Available: https://www.intel.com/content/dam/www/programmable/us/en/pdfs/literature/wp/wp-01222-understanding-peak-floating-point-performance-claims.pdf

[157] H. Giefers and et al., "Accelerating Arithmetic Kernels with Coherent Attached FPGA Coprocessors," in *Proceedings of the 2015 Design, Automation & Test in Europe Conference & Exhibition*, ser. DATE '15. San Jose, CA, USA: EDA Consortium, 2015, p. 1072–1077.

[158] B. Sukhwani *et al.*, "ConTutto – A Novel FPGA-based Prototyping Platform Enabling Innovation in the Memory Subsystem of a Server Class Processor," in *2017 50th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, Oct 2017, pp. 15–26.

[159] A. Data, "ADM-PCIE-9V3," https://www.alpha-data.com/dcp/products.php?product=adm-pcie-9v3.

[160] ——, "ADM-PCIE-9H7," https://www.alpha-data.com/dcp/products.php?product=adm-pcie-9h7.

[161] A. J. Awan, M. Brorsson, V. Vlassov, and E. Ayguade, "Micro-Architectural Characterization of Apache Spark on Batch and Stream Processing Workloads," in *2016 IEEE International Conferences on Big Data and Cloud Computing (BDCloud), Social Computing and Networking (SocialCom), Sustainable Computing and Communications (SustainCom) (BDCloud-SocialCom-SustainCom)*, 2016, pp. 59–66.

[162] V. Dang and et al., "GPU cluster implementation of FMM-FFT for large-scale electromagnetic problems," *IEEE Antennas and Wireless Propagation Letters*, vol. 13, pp. 1259–1262, 2014.

[163] C. L. Yu *et al.*, "FPGA architecture for 2d discrete Fourier transform based on 2d decomposition for large-sized data," *Journal of Signal Processing Systems*, vol. 64, no. 1, pp. 109–122, 2011.

[164] B. Akin *et al.*, "Memory bandwidth efficient two-dimensional fast Fourier transform algorithm and implementation for large problem sizes," *Proceedings of the 2012 IEEE 20th International Symposium on Field-Programmable Custom Computing Machines, FCCM 2012*, pp. 188–191, 2012.

[165] L. Polok and P. Smrz, "Increasing Double Precision Throughput on NVIDIA Maxwell GPUs," in *Proceedings of the 24th High Performance Computing Symposium*, ser. HPC '16. San Diego, CA, USA: Society for Computer Simulation International, 2016. [Online]. Available: https://doi.org/10.22360/SpringSim.2016.HPC.032

[166] I. C. Society, "IEEE Standard for Binary Floating-Point Arithmetic," *ANSI/IEEE Std 754-1985*, pp. 1–20, 1985.

[167] ——, "IEEE Standard for Binary Floating-Point Arithmetic (revision 2008)," *ANSI/IEEE Std 754-1985*, pp. 1–20, 2008.

[168] G. Tagliavini, S. Mach, D. Rossi, A. Marongiu, and L. Benin, "A transprecision floating-point platform for ultra-low power computing," in *2018 Design, Automation Test in Europe Conference Exhibition (DATE)*, 2018, pp. 1051–1056.

[169] A. V. Cueva, "Code Sample: Intel® Deep Learning Boost New Deep Learning Instruction bfloat16 - Intrinsic Functions," https://software.intel.com/content/www/us/en/develop/articles/intel-deep-learning-boost-new-instruction-bfloat16.html?wapkw=bfloat16, 2020.

[170] S. Wang and P. Kanwar, "BFloat16: The secret to high performance on Cloud TPUs," https://cloud.google.com/blog/products/ai-machine-learning/bfloat16-the-secret-to-high-performance-on-cloud-tpus, 2020.

[171] Seehyun Kim, Ki-Il Kum, and Wonyong Sung, "Fixed-point optimization utility for C and C++ based digital signal processing programs," *IEEE Transactions on Circuits and Systems II: Analog and Digital Signal Processing*, vol. 45, no. 11, pp. 1455–1464, 1998.

[172] M. Nguyen, N. Serafin, and J. C. Hoe, "Partial Reconfiguration for Design Optimization," in *2020 30th International Conference on Field-Programmable Logic and Applications (FPL)*, 2020, pp. 328–334.

[173] L. Qiao, G. Luo, W. Zhang, and M. Jiang, "FPGA Acceleration of Ray-Based Iterative Algorithm for 3D Low-Dose CT Reconstruction," in *2020 30th International Conference on Field-Programmable Logic and Applications (FPL)*, 2020, pp. 98–102.

[174] G. Flegar, F. Scheidegger, V. Novaković, G. Mariani, A. E. Tom´ s, A. C. I. Malossi, and E. S. Quintana-Ortí, "FloatX: A C++ Library for Customized Floating-Point Arithmetic," *ACM Transactions on Mathematical Software (TOMS)*, vol. 45, no. 4, Dec. 2019. [Online]. Available: https://doi.org/10.1145/3368086

[175] Gustafson and Yonemoto, "Beating Floating Point at Its Own Game: Posit Arithmetic," *Supercomput. Front. Innov.: Int. J.*, vol. 4, no. 2, p. 71–86, Jun. 2017. [Online]. Available: https://doi.org/10.14529/jsfi170206

[176] E. T. L. Omtzigt, P. Gottschling, M. Seligman, and W. Zorn, "Universal Numbers Library: design and implementation of a high-performance reproducible number systems library," *arXiv:2012.11011*, 2020.

[177] ——, "Universal: a header-only C++ template library for universal number arithmetic," https://github.com/stillwater-sc/universal.

[178] L. Forget, Y. Uguen, and F. De Dinechin, "Hardware cost evaluation of the posit number system," in *Compas'2019 - Conférence d'informatique en Parallélisme, Architecture et Système*, Anglet, France, Jun. 2019, pp. 1–7. [Online]. Available: https://hal.inria.fr/hal-02131982

[179] W.-F. Chiang, M. Baranowski, I. Briggs, A. Solovyev, G. Gopalakrishnan, and Z. Rakamarić, "Rigorous Floating-Point Mixed-Precision Tuning," *SIGPLAN Not.*, vol. 52, no. 1, p. 300–315, Jan. 2017. [Online]. Available: https://doi.org/10.1145/3093333.3009846

[180] M. O. Lam and J. K. Hollingsworth, "Fine-grained floating-point precision analysis," *The International Journal of High Performance Computing Applications*, vol. 32, pp. 231 – 245, 2018.

[181] A. Borghesi, G. Tagliavini, M. Lombardi, L. Benini, and M. Milano, "Combining Learning and Optimization for Transprecision Computing," in *Proceedings of the 17th ACM International Conference on Computing Frontiers*, ser. CF '20. New York, NY, USA: Association for Computing Machinery, 2020, p. 10–18. [Online]. Available: https://doi.org/10.1145/3387902.3392615

[182] S. Cherubin, D. Cattaneo, M. Chiari, A. D. Bello, and G. Agosta, "TAFFO: Tuning Assistant for Floating to Fixed Point Optimization," *IEEE Embedded Systems Letters*, vol. 12, no. 1, pp. 5–8, 2020.

[183] R. DiCecco, L. Sun, and P. Chow, "FPGA-based training of convolutional neural networks with a reduced precision floating-point library," in *2017 International Conference on Field Programmable Technology (ICFPT)*, 2017, pp. 239–242.

[184] D. B. Thomas, “Templatised Soft Floating-Point for High-Level Synthesis,” in *2019 IEEE 27th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, 2019, pp. 227–235.

[185] ——, “Compile-Time Generation of Custom-Precision Floating-Point IP using HLS Tools,” in *2019 IEEE 26th Symposium on Computer Arithmetic (ARITH)*, 2019, pp. 192–193.

[186] “HLS Custom-Precision Floating-Point Library,” https://github.com/dicecco1/fpga_cpfp.

[187] D. B. Thomas, “Templatised Soft Floating-Point for High-Level Synthesis,” https://github.com/template-hls/template-hls-float.

[188] F. de Dinechin, “Reflections on 10 years of FloPoCo,” in *26th IEEE Symposium of Computer Arithmetic (ARITH-26)*, Jun. 2019.

[189] “WSClean,” https://gitlab.com/aroffringa/wsclean.

[190] ASTRON. [Online]. Available: https://support.astron.nl/LOFARImagingCookbook/tutorial.html

[191] “LOFAR Long Term Archive,” https://lta.lofar.eu/.

[192] “Casacore,” https://github.com/casacore/casacore.

[193] “Dysco,” https://github.com/aroffringa/dysco.

[194] “Kstars,” https://docs.kde.org/trunk5/en/extragear-edu/kstars/tool-fitsviewer.html.

[195] G. Flegar, F. Scheidegger, V. Novaković, G. Mariani, A. E. Tomás, A. C. I. Malossi, and E. S. Quintana-Ortí, “FloatX,” https://github.com/oprecomp/FloatX.

[196] Z. Wang, A. Bovik, H. Sheikh, and E. Simoncelli, “Image quality assessment: from error visibility to structural similarity,” *IEEE Transactions on Image Processing*, vol. 13, no. 4, pp. 600–612, 2004.

[197] O. S. Faragallah, H. El-Hoseny, W. El-Shafai, W. A. El-Rahman, H. S. El-Sayed, E.-S. M. El-Rabaie *et al.*, “A Comprehensive Survey Analysis for Present Solutions of Medical Image Fusion and Future Directions,” *IEEE Access*, vol. 9, pp. 11 358–11 371, 2021.

[198] Intel, “Enabling High-Performance Floating-Point Designs,” https://www.intel.com/content/dam/www/programmable/us/en/pdfs/literature/wp/wp-01267-fpgas-enable-high-performance-floating-point.pdf, 2020.

[199] ——, “Intel Stratix 10 Variable Precision DSP Blocks Overview ,” https://www.intel.com/content/dam/www/programmable/us/en/pdfs/literature/wp/wp-01267-fpgas-enable-high-performance-floating-point.pdf.

[200] Xilinx, “Large FPGA Methodology Guide,” https://www.xilinx.com/support/documentation/sw_manuals/xilinx14_7/ug872_largefpga.pdf.

[201] ——, “Xilinx Alveo U50,” https://www.xilinx.com/products/boards-and-kits/alveo/u50.html.

[202] ——, “Vitis Unified Software Development Platform 2020.2 Documentation,” https://www.xilinx.com/html_docs/xilinx2021_1/vitis_doc/devckernels.html.

[203] “PowerSensor Library,” https://gitlab.com/astron-misc/libpowersensor.

[204] “Nvidia 387.12 breaks power reading in nvidia-smi,” https://forums.developer.nvidia.com/t/nvidia-387-12-breaks-power-reading-in-nvidia-smi/53946/21.

[205] S. Sarangi and B. Baas, “DeepScaleTool: A Tool for the Accurate Estimation of Technology Scaling in the Deep-Submicron Era,” in *2021 IEEE International Symposium on Circuits and Systems (ISCAS)*, 2021, pp. 1–5.

[206] “clpeak,” https://github.com/krrishnarraj/clpeak.

[207] Samsung. [Online]. Available: https://www.samsung.com/us/sas/Business/Foundry14nm

[208] TSMC. [Online]. Available: https://www.tsmc.com/english

[209] GlobalFoundries. [Online]. Available: https://gf.com/

[210] E. Calore and S. F. Schifano, "Performance assessment of fpgas as hpc accelerators using the fpga empirical roofline," in *2021 31st International Conference on Field-Programmable Logic and Applications (FPL)*, 2021, pp. 83–90.

[211] Xilinx, "Performance and Resource Utilization for Floating-point v7.1." [Online]. Available: https://gf.com/

[212] "HLS Pragmas," https://www.xilinx.com/html_docs/xilinx2021_1/vitis_doc/hls_pragmas.html#sxx1504034358866.

[213] Y. Hu, Y. Du, E. Ustun, and Z. Zhang, "GraphLily: Accelerating Graph Linear Algebra on HBM-Equipped FPGAs," *Power (Watts)*, vol. 268, no. 182, p. 49, 2021.

[214] "75220 - Alveo U50 - Limited number of HBM2 pseudo-channel ports," https://support.xilinx.com/s/article/75220?language=en_US.

[215] G. Daughtry, "Top 5 Timing Closure Techniques," https://www.xilinx.com/publications/prod_mktg/club_vivado/presentation-2015/paris/Xilinx-TimingClosure.pdf.

[216] N. Brown, "Weighing Up the New Kid on the Block: Impressions of using Vitis for HPC Software Development," in *2020 30th International Conference on Field-Programmable Logic and Applications (FPL)*, 2020, pp. 335–340.

[217] N. Zhang, X. Chen, and N. Kapre, "RapidLayout: Fast Hard Block Placement of FPGA-optimized Systolic Arrays using Evolutionary Algorithms," 2020.

[218] GPUOpen, "AMD RDNA™ 2 Performance Guide - GPUOpen."

[219] C. Wu, M. Wang, X. Chu, K. Wang, and L. He, "Low Precision Floating-point Arithmetic for High Performance FPGA-based CNN Acceleration," 2020.

[220] P. Colangelo, N. Nasiri, E. Nurvitadhi, A. Mishra, M. Margala, and K. Nealis, "Exploration of Low Numeric Precision Deep Learning Inference Using Intel® FPGAs," in *2018 IEEE 26th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, 2018, pp. 73–80.

[221] E. Vermij, L. Fiorin, R. Jongerius, C. Hagleitner, and K. Bertels, "Challenges in exascale radio astronomy: Can the SKA ride the technology wave?" *The International Journal of High Performance Computing Applications*, vol. 29, no. 1, pp. 37–50, 2015. [Online]. Available: https://doi.org/10.1177/1094342014549059

[222] B. Veenboer, M. Petschow, and J. W. Romein, "Image-Domain Gridding on Graphics Processors," in *2017 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, 2017, pp. 545–554.

[223] J. Hou, Y. Zhu, S. Du, S. Song, and Y. Song, "FPGA-Based Scale-Out Prototyping of Degridding Algorithm for Accelerating Square Kilometre Array Telescope Data Processing," *IEEE Access*, vol. 8, pp. 15 586–15 597, 2020.

[224] N. M. Gürel, K. Kara, A. Stojanov, T. Smith, T. Lemmin, D. Alistarh *et al.*, "Compressive Sensing Using Iterative Hard Thresholding With Low Precision Data Representation: Theory and Applications," *IEEE Transactions on Signal Processing*, vol. 68, pp. 4268–4282, 2020.

[225] M. Seznec, N. Gac, A. Ferrari, and F. Orieux, "A Study on Convolution using Half-Precision Floating-Point Numbers on GPU for Radio Astronomy Deconvolution," in *2018 IEEE International Workshop on Signal Processing Systems (SiPS)*, 2018, pp. 170–175.

[226] T. Nguyen, S. Williams, M. Siracusa, C. MacLean, D. Doerfler, and N. J. Wright, "The performance and energy efficiency potential of fpgas in scientific computing," in *2020 IEEE/ACM Performance Modeling, Benchmarking and Simulation of High Performance Computer Systems (PMBS)*, 2020, pp. 8–19.

[227] Y.-k. Choi, Y. Chi, W. Qiao, N. Samardzic, and J. Cong, "HBM Connect: High-Performance HLS Interconnect for FPGA HBM," in *The 2021 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, ser. FPGA '21. New York, NY, USA: Association for Computing Machinery, 2021, p. 116–126. [Online]. Available: https://doi.org/10.1145/3431920.3439301

[228] B. Gaide, D. Gaitonde, C. Ravishankar, and T. Bauer, "Xilinx Adaptive Compute Acceleration Platform: Versal<sup>TM</sup> Architecture," in *Proceedings of the 2019 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, ser. FPGA '19. New York, NY, USA: Association for Computing Machinery, 2019, p. 84–93. [Online]. Available: https://doi.org/10.1145/3289602.3293906

[229] Samsung. PIM. [Online]. Available: https://semiconductor.samsung.com/insights/technology/pim/

[230] UPMEM. UPMEM - Technology. [Online]. Available: https://www.upmem.com/technology/

[231] Intel. Performance Index - ISC 2022. [Online]. Available: https://edc.intel.com/content/www/cn/zh/products/performance/benchmarks/isc-2022/

[232] T. Shimizu, "Supercomputer fugaku: Co-designed with application developers/researchers," in *2020 IEEE Asian Solid-State Circuits Conference (A-SSCC)*, 2020, pp. 1–4.

[233] NVIDIA. NVIDIA GRACE CPU. [Online]. Available: https://www.nvidia.com/en-us/data-center/grace-cpu/

[234] Xilinx. Alveo U280 Data Center Accelerator Card. [Online]. Available: https://www.xilinx.com/products/boards-and-kits/alveo/u280.html

[235] T. Jia, Y. Ju, R. Joseph, and J. Gu, "Ncpu: An embedded neural cpu architecture on resource-constrained low power devices for real-time end-to-end performance," in *2020 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2020, pp. 1097–1109.

[236] B. W. Mezger, D. A. Santos, L. Dilillo, C. A. Zeferino, and D. R. Melo, "A survey of the risc-v architecture software support," *IEEE Access*, vol. 10, pp. 51 394–51 411, 2022.

[237] A. Computing. High Performance Scalable Sustainable Ampere® Cloud Native Processors. [Online]. Available: https://amperecomputing.com/processors/ampere-altra/

[238] NVIDIA. NVIDIA Omniverse. [Online]. Available: https://www.nvidia.com/en-us/omniverse/

[239] A. Li, S. L. Song, E. Brugel, A. Kumar, D. Chavarria-Miranda, and H. Corporaal, "X: A comprehensive analytic model for parallel machines," in *2016 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. IEEE, 2016, pp. 242–252.

[240] S. De, S. Mohamed, K. Bimpisidis, D. Goswami, T. Basten, and H. Corporaal, "Approximation trade offs in an image-based control system," in *2020 Design, Automation Test in Europe Conference Exhibition (DATE)*, 2020, pp. 1680–1685.

[241] S. De, J. Huisken, and H. Corporaal, "An automated approximation methodology for arithmetic circuits," in *2019 IEEE/ACM International Symposium on Low Power Electronics and Design (ISLPED)*, 2019, pp. 1–6.

[242] S. Ullah, T. D. A. Nguyen, and A. Kumar, "Energy-efficient low-latency signed multiplier for fpga-based hardware accelerators," *IEEE Embedded Systems Letters*, vol. 13, no. 2, pp. 41–44, 2021.

[243] S. Ullah, H. Schmidl, S. S. Sahoo, S. Rehman, and A. Kumar, "Area-optimized accurate and approximate softcore signed multiplier architectures," *IEEE Transactions on Computers*, vol. 70, no. 3, pp. 384–392, 2021.

[244] Z. E. Mamaghani, S. Ullah, and A. Kumar, "Simdive: Approximate simd soft multiplier-divider for fpgas with tunable accuracy," in *30th 2020 ACM Great Lakes Symposium on VLSI (GLSVLSI)*, September 2020.

[245] T. Cheng, Y. Masuda, J. Chen, J. Yu, and M. Hashimoto, "Logarithm-approximate floating-point multiplier is applicable to power-efficient neural network training," *Integration*, vol. 74, pp. 19 – 31, 2020. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0167926019305826

[246] Romein, John W., "The tensor-core correlator," *A&A*, vol. 656, p. A52, 2021. [Online]. Available: https://doi.org/10.1051/0004-6361/202141896

[247] J. A. Högbom, “Aperture Synthesis with a Non-Regular Distribution of Interferometer Baselines,” *Astronomy & Astrophysics, Supplement*, vol. 15, p. 417, Jun. 1974.

[248] L. Connor, K. L. Bouman, V. Ravi, and G. Hallinan, “Deep radio-interferometric imaging with POLISH: DSA-2000 and weak lensing,” *Monthly Notices of the Royal Astronomical Society*, vol. 514, no. 2, pp. 2614–2626, 05 2022. [Online]. Available: https://doi.org/10.1093/mnras/stac1329

[249] “Xilinx’s Arbitrary Precision Data Types Library,” https://www.xilinx.com/html_docs/xilinx2020_2/vitis_doc/ogi1585574074269.html.

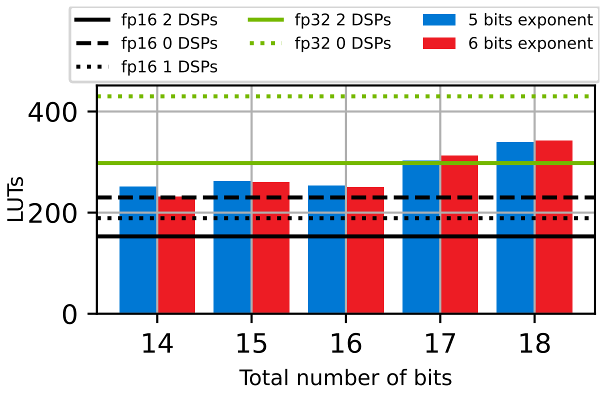
Acronyms

AI	Artificial Intelligence.
ALU	Arithmetic Logic Unit.
API	Application Programming Interface.
BRAM	Block RAM.
CLB	Configurable Logic Block.
CPI	Clock Per Instruction.
CPU	Central Processing Unit.
CUDA	Compute Unified Device Architecture.
BBLP	Basic-Block Level Parallelism.
DDR	Double Data Rate.
DLP	Data Level Parallelism.
DRAM	Dynamic Random Access Memory.
DSE	Design Space Exploration.
DSP	Digital Signal Processing.
DTR	Data Temporal Reuse.
EDP	Energy Delay Product.
FMA	Fused Multiply and Accumulate.
FFT	Fast Fourier Transform.
FPGA	Field Programmable Gate Array.
GPC	Graphic Processing Cluster.
GPU	Graphics Processing Unit.
HBM	High Bandwidth Memory.
HDL	Hardware Description Language.
HMC	Hybrid Memory Cube.
HPC	High Performance Computing.
IDG	Image Domain Gridding.
ILP	Instruction Level Parallelism.
ISA	Instruction Set Architecture.
IR	Intermediate Representation.
JSON	JavaScript Object Notation.
LLVM	Low Level Virtual Machine.
LOFAR	Low Frequency ARray.
LSU	Load Store Unit.
LTA	Long Term Archive.
LUT	Look-Up Table.

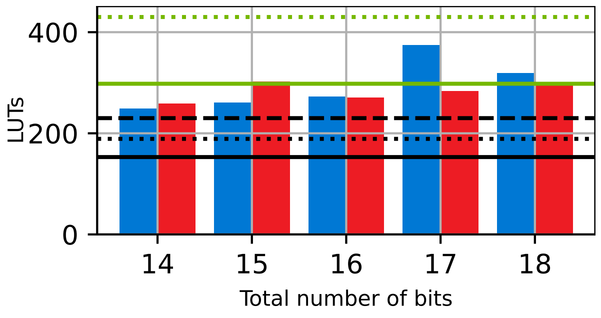
MAC	Multiply and ACcumulate.
ML	Machine Learning.
MPI	Message Passing Interface.
MSE	Mean Squared Error.
NaN	Not a Number.
NMC	Near Memory Computing.
OpenMP	Open Multiprocessing.
PAPI	Performance Application Profiling Interface.
PCA	Principal Component Analysis.
PE	Processing Elements.
PIM	Processing In Memory.
PISA	Platform Independent Software Analysis.
PMU	Performance Monitoring Unit.
PSNR	Peak Signal to Noise Ration.
RF	Random Forest.
SDRAM	Synchronous Dynamic Random Access Memory.
SIMD	Single Instruction Multiple Data.
SLR	Super Logic Region.
SSA	Static Single Assignment.
SSIM	Structural Similarity Index Measure.
TSV	Through-Silicon Via.
URAM	Ultra RAM.

A HLS custom floating-point library comparison

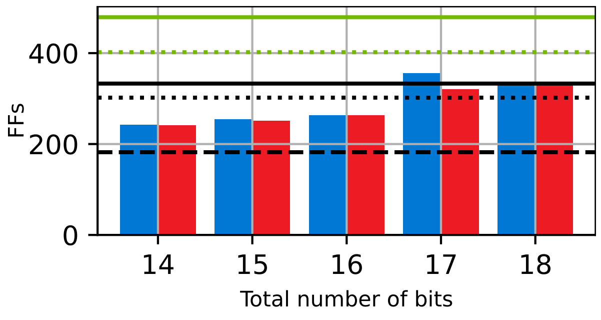
The main custom floating-point libraries for High-Level-Synthesis are the custom-precision floating-point (cpfp) [186], and the templatised soft floating-point for high-level synthesis (thls) [187]. Both the libraries are based on the Xilinx arbitrary precision library [249]. While cpfp supports homogeneous custom floating-point, thls enables heterogeneous custom floating-point, which may be helpful for fine-grained mixed-precision or compute operations at higher precision and then truncate the result. Since we do not not need heterogeneous operators for this work the cpfp library would be sufficient. However, we evaluate the resource usage of the two libraries for the adder (see *Figure A.1*) and multiplier operator (see *Figure A.2*) to motivate our choice of thls over cpfp.
Both the operators are convenient compared to single-precision floating-point for the presented precision, including the one employed in our highest performing accelerator (6 bits exponent and 11 bits mantissa, with a total of 18 bits). Unlike the adder operator, the multiplier operator, when considering the same precision and number of DSPs (1), has similar resource usage (less in the case of thls) as half-precision. Overall thls has reduced LUTs, e.g. *Figure A.2b* vs *Figure A.2a*, and FFs, e.g. *Figure A.2d* vs *Figure A.2c*, usage compared to cpfp. This analysis thus motivate the usage of thls over cpfp.



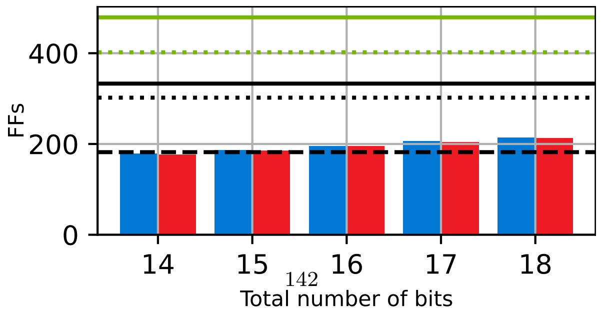
(a) LUTs usage cpfp adder.



(b) LUTs usage thls adder.

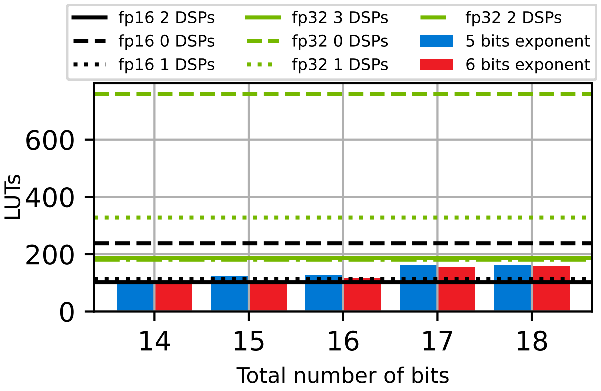


(c) FFs usage cpfp adder.

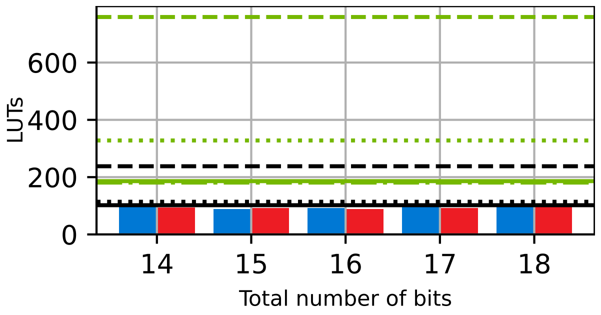


(d) FFs usage thls adder.

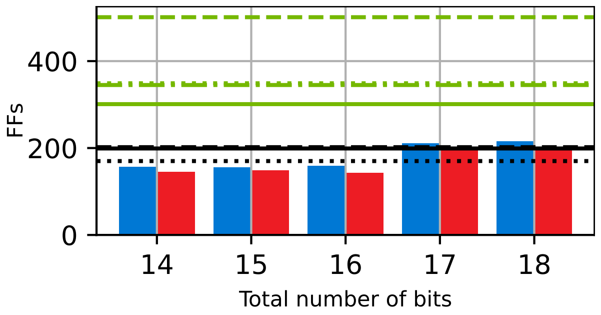
Figure A.1: *Custom floating-point adder (not using DSP) resource usage for the cpfp and thls libraries.*



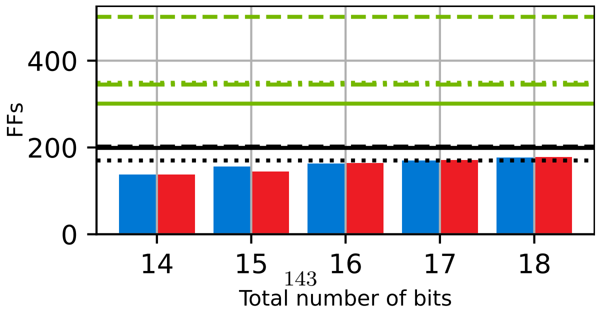
(a) LUTs usage cpfp multiplier.



(b) LUTs usage thls multiplier.



(c) FFs usage cpfp multiplier.



(d) FFs usage thls multiplier.

Figure A.2: *Custom floating-point multiplier (1 DSP) resource usage for the cpfp and thls libraries.*

B
List of Publications

1. Stefano Corda, Bram Veenboer, Emma Tolley, **PMT: Power Measurement Toolkit**, SC22 (HUST)

2. Stefano Corda, Bram Veenboer, Ahsan Javed Awan, John W Romein, Roel Jordans, Akash Kumar, Albert-Jan Boonstra, Henk Corporaal, **Reduced-Precision Acceleration of Radio-Astronomical Imaging on Reconfigurable Hardware**, IEEE Access 2022

3. Stefano Corda, Madhurya Kumaraswamy, Ahsan Javed Awan, Roel Jordans, Akash Kumar, Henk Corporaal, **NMPO: Near-Memory Computing Profiling and Offloading**, DSD 2021 (**Best Student Paper Award**)

4. Stefano Corda, Bram Veenboer, Ahsan Javed Awan, Akash Kumar, Roel Jordans, Henk Corporaal, **Near memory acceleration on high resolution radio astronomy imaging**, MECO 2020

5. Kanishkan Vadivel, Lorenzo Chelini, Ali BanaGozar, Gagandeep Singh, Stefano Corda, Roel Jordans, Henk Corporaal, **TDO-CIM: Transparent Detection and Offloading for Computation In-Memory**, DATE 2020

6. Gagandeep Singh, Lorenzo Chelini, Stefano Corda, Ahsan Javed Awan, Sander Stuijk, Roel Jordans, Henk Corporaal, Albert-Jan Boonstra, **Near-memory computing: Past, present, and future**, MICPRO 2019

7. Stefano Corda, Gagandeep Singh, Ahsan Javed Awan, Roel Jordans, Henk Corporaal, **Platform Independent Software Analysis for Near Memory Computing**, DSD 2019

8. Gagandeep Singh, Juan Gómez-Luna, Giovanni Mariani, Geraldo F Oliveira, Stefano Corda, Sander Stuijk, Onur Mutlu, Henk Corporaal, **NAPEL: Near-Memory computing application Performance prediction via Ensemble Learning**, DAC 2019

9. Stefano Corda, Gagandeep Singh, Ahsan Javed Awan, Roel Jordans, Henk Corporaal, **Memory and Parallelism Analysis Using a Platform-Independent Approach**, SCOPES 2019

10. Jan Van Lunteren, Ronald Luijten, Dionysios Diamantopoulos, Florian Auernhammer, Christoph Hagleitner, Lorenzo Chelini, Stefano Corda, Gagandeep Singh, **Coherently attached programmable near-memory acceleration platform and its application to stencil processing**, DATE 2019

11. Gagandeep Singh, Lorenzo Chelini, Stefano Corda, Ahsan Javed Awan, Sander Stuijk, Roel Jordans, Henk Corporaal, Albert-Jan Boonstra, **A review of near-memory computing architectures: Opportunities and challenges**, DSD 2018

C
CV

Stefano Corda was born on December 30th, 1993, in Moncalieri, Italy. He received a bachelor's degree in Electrical and Electronic Engineering from the University of Cagliari (Italy) in 2015 with the thesis "Power Saving in reconfigurable dataflow-based systems". He graduated in Electronic Engineering in 2017 at the same University with the thesis "Extension and optimization of a Co-processor for Convolutional Neural Network". He joined the Electronic System (ES) group at the Eindhoven University of Technology in 2017 as a Ph.D. candidate funded by a Marie Skłodowska-Curie EID grant under the supervision of Professor Henk Corporaal and Dr. Roel Jordans. During his Ph.D., he visited IBM Research Zürich (Switzerland). He also visited as Guest Ph.D. Dresden Technical University (Germany) under the supervision of Professor Akash Kumar. His Ph.D. research is described in this thesis. In 2022 he joined the Scientific IT and Application Support (SCITAS) group at the École Polytechnique Fédérale de Lausanne (EPFL) as High-Performance Computing (HPC) Application Expert to work on the HPC Hardware/Software Co-design for Square Kilometre Array (SKA) project.