Technische Universität München
TUM School of Computation, Information and Technology

Design Space Exploration for Approximate Image Processing on FPGAs

Manu Manuel

Vollständiger Abdruck der von der TUM School of Computation, Information and Technology der Technischen Universität München zur Erlangung des akademischen Grades eines

Doktors der Ingenieurwissenschaften (Dr.-Ing.)

genehmigten Dissertation.

Die Dissertation wurde am 17.07.2023 bei der Technischen Universität München eingereicht und durch die TUM School of Computation, Information and Technology am 29.11.2023 angenommen.

Acknowledgment

I would like to take this opportunity to express my deepest gratitude to all those who supported me during this work. First, I would like to thank my supervisor, Prof. Dr.-Ing. Walter Stechele, who continuously supported me throughout this journey. He was not just my supervisor but rather a mentor, motivator, and solution finder to me. Thank you, Walter, for your kindness in addressing all my concerns, regardless of the subjects and their relevance, and I was delighted to work with you. I would then like to express my sincere thanks to my research partners, Arne Kreddig from SmartRay GmbH and Simon Conrady from Arnold & Richter Cine Technik (ARRI), for your selfless collaboration and for being one another during the whole journey. Our continuous discussions and your supports were invaluable in learning and growing. I couldn't imagine better partners than you for this challenging work.

I would like to acknowledge the management of the Chair of Integrated Systems, my fellow colleagues, and the students with whom I cooperated, for your support and all the good moments, especially Dr. Ir. Anh Vu Doan for guiding me in genetic algorithms, Anmol Surhonne for all our technical and non-technical discussions, and Franz Biersack for helping me with all the translations. In addition, I would like to thank Dr.-Ing. Daniel Müller-Gritschneder for your efforts in reviewing and examining this work, and Prof. Dr.-Ing. Gerhard Rigoll for chairing the examination committee. I would like to acknowledge the Bavarian Ministry of Economic Affairs, Regional Development and Energy (Grant No. IUK574) and Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) Project No. 146371743, TRR 89: Invasive Computing, for the financial assistance to carry out this research successfully.

I want to express my heartfelt thanks to my wife Nelsy Ann Joseph and my sons David Manuel and Daniel Manuel, who shared all my ups and downs and offered unwavering support during this period. My wife is the backbone of this work by taking all the family responsibilities away from me, which allowed me to concentrate on my research. I want to convey my deepest gratitude to my father the late Manuel Thomas, my mother Jessy Manuel, and my sister Rose Maria Manuel, for your life lessons and sacrifices that helped me to grow personally and professionally. Thanks to my parents-in-law, grandparents, other family members, and friends for your love, support, and encouragement throughout my life journey. Above all, I would like to thank my God for everything you have given me.

Abstract

The ever-growing demand for increased computational power due to exponential data growth in recent years pushes the capabilities of traditional computational technology to their limits. To reduce the consequent gap between the processing demands and the computational abilities, many studies have been proposed over the years, redefining the principles of conventional computations. The research field of approximate computing introduces a quality-aware computation in error-resilient applications such as image or signal processing by safely trading off the application quality for resource savings on computing systems. Therefore, an additional quality-threshold parameter determines the boundary of acceptable quality loss and redefines the concept of correctness in approximate computing applications. This thesis presents a comprehensive survey and classification of various state-of-the-art approximate computing techniques that trade off the application quality for better hardware resources. In real-world applications with distinct system components, a careful selection and combination of multiple of these single-purpose approximation techniques tailored to the target application can exploit the maximum benefits from approximate computing. However, accuracy configurable parameters exposed from all these approximate techniques in an application lead to a non-trivial multi-objective parameter optimization task, where a global optimization on joint parametrization is necessary to include error propagation effect between system components. Various approximate computing design flows are proposed to address this problem. This thesis also presents a survey on such design flows that combine multiple approximations in an application and determine the quality-resource trade-off.

Addressing the shortcomings of state-of-the-art approximate computing design flows, this thesis proposes AxCGA: a design space exploration (DSE) framework for approximate computing that combines parametrizable approximations in real-world field programmable gate array (FPGA)-based image processing applications and explores the resulting design space to determine the quality-resource trade-off. In AxCGA, a data flow graph based approach is adapted for combining approximations in a target application, and the DSE is designed based on genetic algorithm and non-dominated sorting genetic algorithm-II (NSGA-II) selection. A pre-characterized library of approximate components is used in AxCGA with fast and accurate models for the fitness estimation of each parameter combination during DSE. Different components in AxCGA, such as genetic encoding, initial population formation, genetic operations to form the offspring population, hyperparameter settings, selection mechanism, and stop condition are adapted to support the generality and adaptability to different approximate computing applications. Additionally, a novel approach adaptively provides genetic algorithm parameters during the DSE and can further exclude the application-specific hyperparameter optimization. The usability of the AxCGA framework is demonstrated on two different applications, such as RGB to $YCbCr$ conversion and display rendering application, and the DSE experiments resulted in well-distributed Pareto fronts where a designer can select desired configurations for implementation. An average hypervolume comparison shows that the AxCGA outperforms the state-of-the-art autoAx DSE, and an additional performance comparison between adaptive and non-adaptive AxCGA shows a slightly better average hypervolume in adaptive approach.

To improve the efficiency of AxCGA, this thesis proposes a novel three-phase region of interest (ROI) based NSGA-II (ROI-NSGA-II) selection that can incorporate quality-threshold into a DSE problem. Based on this quality-threshold and the resources from reference implementation, ROI bounds in both resource and quality dimensions can be specified in approximate computing applications. In ROI-NSGA-II, the first set of points within the desired ROI is identified in the initialization phase. Thereafter, more points are aggregated within the ROI in the second invitation phase. The final set of points is evolved within the ROI by maintaining both convergence and diversity in the third conquest phase. A performance comparison in average hypervolume from the ROI on both case studies shows that ROI-NSGA-II AxCGA outperforms both the NSGA-II AxCGA and autoAx DSE. Additionally, an adaptive and non-adaptive comparison in average hypervolume shows a marginal improvement in adaptive genetic algorithm.

The novel ROI-NSGA-II is verified and validated to show its capabilities of identifying Pareto solutions in two objective Zitzler-Deb-Thiele (ZDT) benchmark problems with concave, convex, and disconnected Pareto shapes and Deb-Thiele-Laumanns-Zitzler (DTLZ) problems with three to ten objectives. A performance comparison with state-of-the-art preference-based R-NSGA-II selection shows that ROI-NSGA-II performs equally or marginally better in two and three objective problems, whereas it outperformed the R-NSGA-II in many objective problems and both the approximate computing case studies.

Zusammenfassung

Die stetig wachsende Anforderung nach höherer Rechenleistung aufgrund des exponentiellen Datenwachstums in den letzten Jahren treibt die traditionelle Rechentechnologie an ihre Leistungsgrenzen. Um die hieraus resultierende Lücke zwischen Rechenanforderungen und der möglichen Rechenleistung zu reduzieren, wurden über die Jahre hinweg viele Studien vorgeschlagen, die die Prinzipien konventioneller Rechentechnologie neu definierten. Das Forschungsgebiet des Approximate Computing stellt eine qualitätsbewusste Berechnung fehlertoleranter Anwendungen, wie Bild- oder Signalverarbeitung vor, indem sorgfältig zwischen der Anwendungsqualität und Ressourceneinsparungen von Rechensystemen abgewogen wird. Zu diesem Zweck definiert ein zusätzlicher Qualitätsschwellwert-Parameter die Grenze des maximal akzeptierbaren Qualitätsverlustes und definiert somit das Konzept von Korrektheit in Anwendungen von Approximate Computing neu. Diese Arbeit präsentiert eine umfassende Recherche und Klassifizierung unterschiedlicher State-of-the-Art-Techniken des Approximate Computing, welche die Anwendungsqualität typischerweise gegen eine bessere Ausnutzung der Hardware-Ressourcen eintauschen. In realen Anwendungen mit unterschiedlichen Systemkomponenten kann eine sorgfältige Auswahl und Kombination mehrerer solcher Approximations-Techniken, jede davon maßgeschneidert für eine einzelne Ziel-Anwendung, die maximalen Vorteile von Approximate Computing ausnutzen. Parameter allerdings, die in ihrer Präzision konfigurierbar sind, und aus all diesen Approximations-Techniken einer Anwendung hervorgehen, führen zu einem nicht-trivialen Parameter-Optimierungs-Problem mit mehreren Zielfunktionen, wo eine globale Optimierung mittels gemeinsamer Parametrisierung notwendig ist, um die Fortplanzungseffekte von Fehlern zwischen Systemkomponenten zu erfassen. Um dieses Problem zu adressieren werden unterschiedliche Design Flows für Approximate Computing vorgeschlagen. Diese Arbeit stellt auch eine Recherche solcher Design Flows vor, die mehrere Approximationen einer Anwendung miteinander verbinden, und sie bestimmt den jeweiligen Qualitäts-Ressourcen-Trade-Off.

Um die Nachteile von State-of-the-Art Approximate Computing Design Flows zu addressieren, stellt diese Arbeit AxCGA vor, ein Design Space Exploration (DSE)-Framework für Approximate Computing, welches parametrisierbare Approximationen in realen Field Programmable Gate Array (FPGA)-basierten Bildverarbeitungsanwendungen miteinander vereint und den sich daraus ergebenden Design-Space analysiert, um den Kompromiss zwischen Qualität und Ressourcen zu bestimmen. In AxCGA wird ein auf einem Datenflussgraphen basierender Ansatz angepasst, um Approximationen in einer Zielanwendung zu verbinden, und das DSE-Design basiert auf der Auswahl eines genetischen Algorithmus und eines genetischen Algorithmus nicht-dominierter Sortierung – II (engl. non-dominated sorting genetic algorithm-II (NSGA-II)). Eine vor-charakterisierte Bibliothek aus Approximierungs-Komponenten wird in AxCGA verwendet, mit schnellen und präzisen Modellen für die Eignungs-Abschätzung jeder Parameter-Kombination während der DSE. Unterschiedliche Komponenten in AxCGA, wie die genetische Kodierung, die intiale Populations-Bildung, genetische Bearbeitungsschritte, um die Nachkommens-Population zu formen, Hyperparameter-Einstellungen, Auswahl-Mechanismen, und die Stopp-Bedingung werden angepasst, um die Allgemeinheit und Anpassbarkeit an unterschiedliche Approximate Computing-Anwendungen zu unterstützen. Zusätzlich stellt ein neuer Ansatz Parameter für den genetischen Algorithmus anpassbar während der DSE zur Verfügung und kann ferner die anwendungsspezifische Hyperparameter-Optimierung vermeiden. Die Benutzbarkeit des AxCGA-Frameworks wird mittels zweier unterschiedlicher Anwendungen demonstriert, wie der RGB-zu-$YCbCr$-Konvertierung und der Anzeige von Render-Anwendungen, und die DSE-Experimente resultierten in eine wohlverteilte Pareto-Front, aus welcher der Designer die gwünschte Konfiguration für die Implementierung wählen kann. Ein Vergleich des durchschnittlichen Hypervolumens zeigt, dass das AxCGA die Stand-der-Technik-autoAx DSE übertrifft. Und ein zusätzlicher Leistungsvergleich zwischen adaptiver und nicht-adaptiver AxCGA zeigt ein etwas besseres durchschnittliches Hypervolumen des adaptiven Ansatzes.

Um die Effizienz von AxCGA zu verbessern, stellt diese Arbeit eine neuartige dreiphasige Region of Interest (ROI)-basierte NSGA-II (ROI-NSGA-II)-Auswahl vor, welche den Qualitäts-Schwellwert in die DSE mit aufnehmen kann. Basierend auf diesem Qualitäts-Schwellwert und den Ressourcen der Referenz-Implementierung können Grenzwerte sowohl in der Ressourcen- als auch in der Qualitäts-Dimension in Approximate Computing Anwendungen spezifiziert werden. In ROI-NSGA-II wird das erste Set an Punkten innerhalb der gewüschten ROI in der Initialisierungs-Phase identifiziert. Anschließend werden während der zweiten Einladungsphase weitere Punkte innerhalb der ROI aggregiert. Das finale Set an Punkten entwickelt sich innerhalb der ROI durch Beibehalten von sowohl Konvergenz als auch Diversität in der dritten Eroberungsphase. Ein Leistungsvergleich des durchschnittlichen Hypervolumens der ROI in beiden Fallstudien zeigt, dass ROI-NSGA-II AxCGA sowohl NSGA-II AxCGA als auch autoAx DSE überlegen ist. Zusätzlich zeigt ein adaptiver wie nicht-adaptiver Vergleich des durchschnittlichen Hypervolumens leichte Verbesserungen des adaptiven genetischen Algorithmus.

Der neuartige ROI-NSGA-II wird verifiziert und validiert, um seine Fähigkeiten darzustellen, Pareto-Lösungen in zwei objektiven Zitzler-Deb-Thiele (ZDT)-Benchmark-Problemen mit konkaven, konvexen und getrennten Pareto-Formen sowie Deb-Thiele-Laumanns-Zitzler (DTLZ)-Problemen mit drei bis zehn Objektiven zu identifizieren. Ein Leistungsvergleich mit einer präferenzbasierten State-of-the-Art-R-NSGA-II-Auswahl zeigt, dass sich ROI-NSGA-II gleich oder leicht besser in zwei und drei objektiven Problemen verhält, wohingegen es der R-NSGA-II in vielen objektiven Problemen und beiden Approximate Computing Fallstudien überlegen ist.

Contents

CHAPTER 1

Introduction

1.1 Motivation

Growing demands for processing power on embedded computing systems outpace the technological improvements in recent years. Since 1965 when Gordon E Moore forecasted that the number of transistors on an integrated circuit would double about every two years [1], Moore's law has driven the transistor technology to shrink its size and pack more functionalities. A decade later, in 1974, the MOSFET scaling by Robert Dennard states that MOSFET power density stays constant as the transistor gets smaller [2], enabling circuits to operate at higher frequencies at the same power consumption. Moore's law combined with Dennard scaling have been driving semiconductor technology over the past decades, increasing the number of transistors in a chip, computing performance, and typical power consumption capabilities. The trends over 50 years of microprocessor features are shown in Figure 1.1

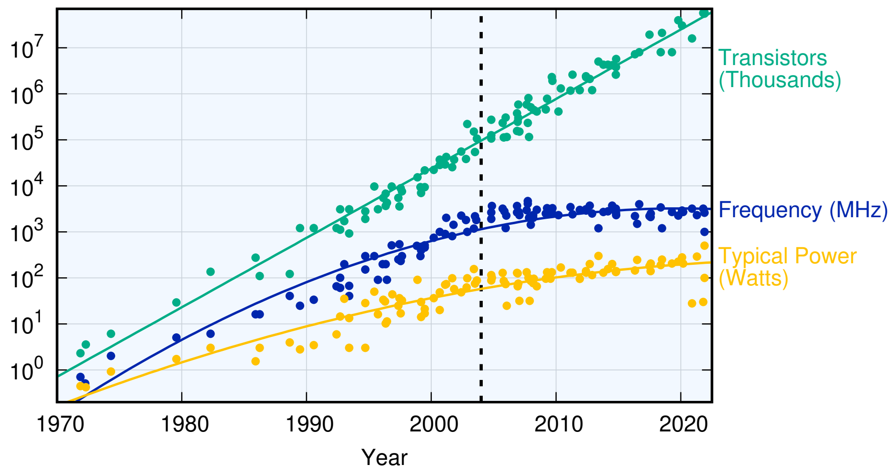


Figure 1.1: The trends over 50 years of microprocessor features (Original data up to 2010 collected by M. Horowitz, F. Labonte, O. Shacham, K. Olukotun, L. Hammond, and C. Batten, and data between 2010-2021 collected by K. Rupp [3]: adapted under CC-BY-4.0)

However, modern computing systems face multiple challenges due to the failure of Dennard scaling from around 2004 [4] and the rising concerns about the failure of Moore's law. Furthermore, the data to be processed exponentially increased over the past decade with advancements in data-intensive applications such as artificial intelligence, augmented reality, machine vision, big data, and internet of things. These challenges together create a gap between the processing power demands and the improvements in technology, especially in mobile devices that have to fulfill significant requirements on power consumption. The deprivation of the computational capabilities of current systems, along with power and energy restrictions, pushes researchers to reconsider traditional computing methodologies.

To address these issues, the research field of approximate computing has gained much traction in recent years. Approximate computing is a digital design paradigm that exploits the inherent error resilience of applications such as image processing or signal processing. The application quality is leveraged in approximate computing for efficient usage of computational resources such as power, area, and performance. An increased degree of approximations in error-resilient applications often facilitates improved benefits over computing resources. Therefore, a quality-resource trade-off analysis is required in approximate computing applications, and Figure 1.2 shows the typical design space exposed in such applications. With an additional *quality-threshold* introduced in approximate computing, approximate designs with resource benefits that meet this minimum quality bound are considered "good enough" designs. In an application, a designer can analyze this trade-off within this region and choose desired designs based on preferences.

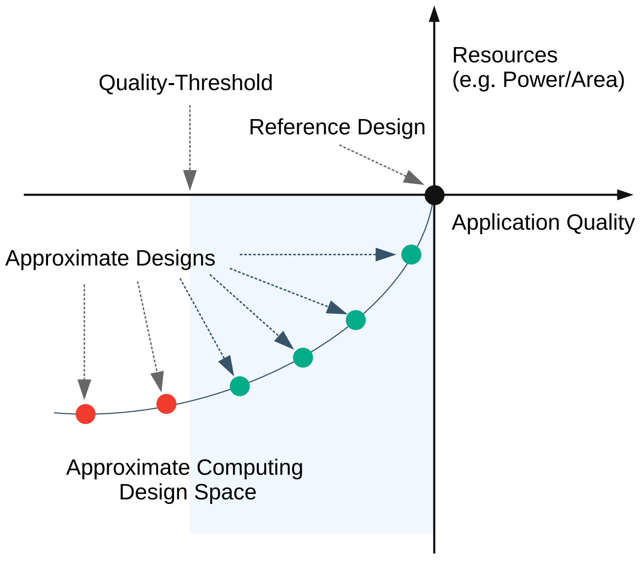


Figure 1.2: Approximate computing design space (adapted from [5])

To apply the approximate computing techniques successfully in an application, the application itself or a part of the application needs to be resilient to the inexactness introduced by the approximations. Therefore, it is essential to identify the applications that can tolerate such inexactness. Recent studies have shown that different types of applications are error-resilient in their nature, inspiring computing methodologies like stochastic computing, probabilistic computing, or approximate computing [6, 7, 8, 9, 10, 11, 12, 13, 14, 15]. Several factors can determine this error resiliency in an application [6, 7, 8, 9], and these factors can be classified based on input characteristics, computing patterns, and output characteristics, as shown in Figure 1.3.

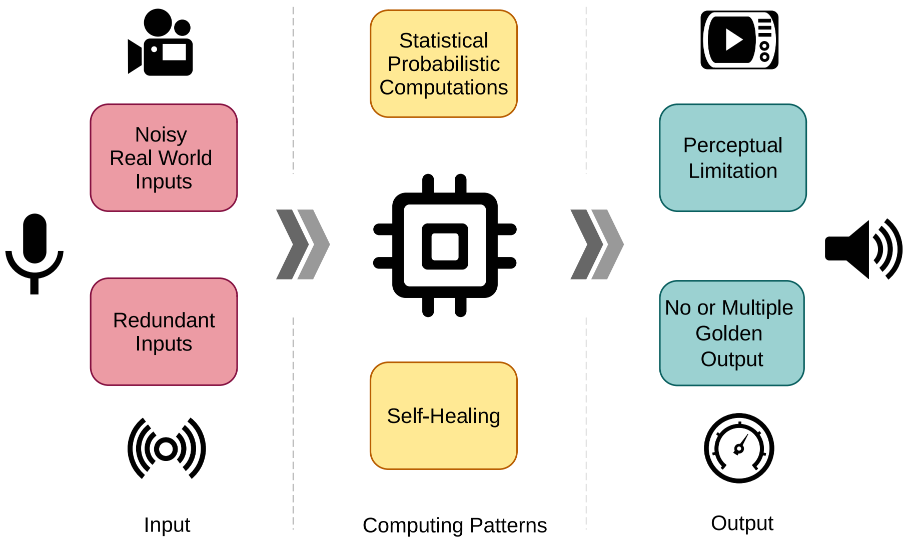
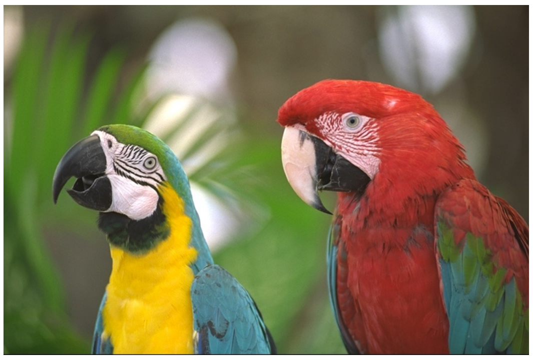


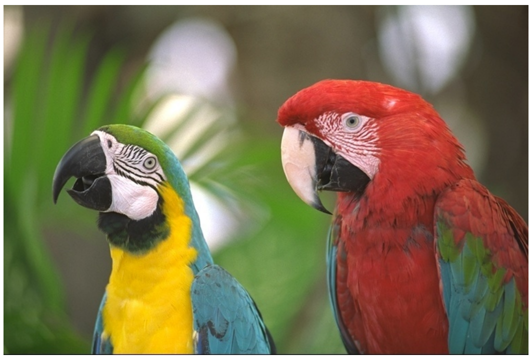
Figure 1.3: Factors affecting inherent error resiliency of the application (inspired from [6, 7, 8, 9])

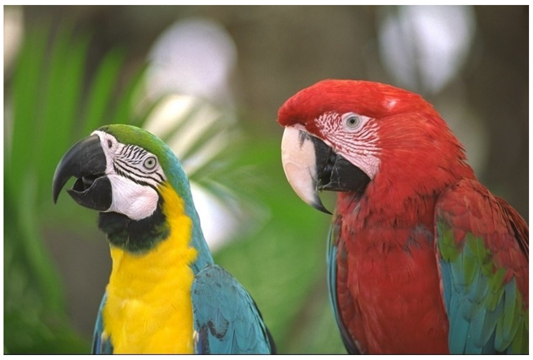
Input characteristics such as noise robustness or redundancy make an application tolerant to errors in the inputs. The error introduced by the approximate computing techniques often resembles the effect of input noise and its propagation over the application components. Therefore, the inherent robustness of an application to such input noise makes it error-resilient. Similarly, input redundancy often makes an application prone to input errors. Secondly, applications with computing patterns that use statistical or probabilistic computations and self-healing applications are also inherently error-resilient. A degradation of application quality from the approximations might recover in the subsequent processing steps in the former case. In contrast, an error introduced by the approximate computation might not be significantly reflected in the final results in the latter case. Finally, the applications that do not have a specific golden output that reflects the reference output quality or have multiple golden outputs, such as cloud search or recommendation systems, are also inherently error-resilient. Furthermore, applications that produce errors beyond the human perceptual limit are also error-resilient. Many image, sound, or video processing applications are error-resilient due to the perceptual limit of humans, and a wide range of output is considered equally acceptable in these applications. Applications with one or more of these characteristics are often suitable candidates for approximate computing.

Processing images with a high dynamic range or spatial-temporal resolution is inevitable in modern image processing applications. However, these applications often have to fulfill performance or power consumption requirements in embedded devices, especially in mobile devices. Since image processing applications often possess multiple error-resilient criteria, such as pixel redundancy among spatial and temporal neighbors, robustness to Gaussian, salt-and-pepper, or shot noises, and perceptual limitations to small output changes, these applications can be considered for approximate computing to achieve better resource utilization and fulfill the processing demands.

The quality of resulting images are either evaluated subjectively with visual experience in motion picture or professional photography cameras or with specific measurements in industrial cameras or other image processing applications. Due to error-resilient characteristics, errors introduced in such applications in a controllable way are often acceptable. For example, images with slightly different peak signal-to-noise ratio (PSNR) produce a similar visual experience to human eyes, as shown in Figure 1.4. In general, many image processing applications possess the properties of inherent application resilience and are often good candidates for approximate computing to fulfill additional resource demands. However, it is important to ensure that the quality-threshold is met in these applications.


(a) Reference image (taken from Kodak image set [16])


(b) Image with PSNR = 50.11 dB


(c) Image with PSNR = 40.20 dB

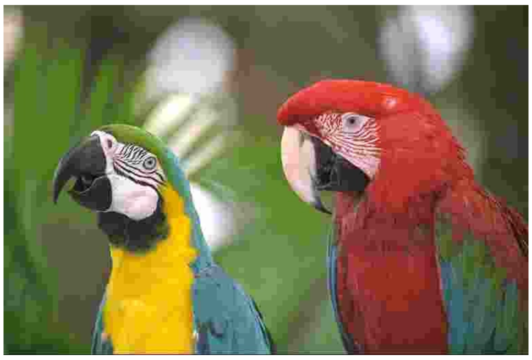

(d) Image with PSNR = 30.89 dB

Figure 1.4: Example 8-bit images with different PSNR levels. In both (b) and (c), the loss of quality is difficult to distinguish with human eyes compared to the reference image in (a). However, in (d), the loss of quality is more evident.

To exploit the error-resiliency in applications for better hardware resource utilization, many approximate computing techniques focusing on different embedded computing hardware such as field-programmable gate array (FPGA), application specific integrated circuit (ASIC) or central processing unit (CPU) have been proposed over the years [9, 13, 14, 15, 17, 18, 19]. These techniques comprise single-purpose methods such as approximate arithmetic units or precision scaling, targeting a specific system component or a single functionality and general-purpose methods such as SASIMI [20] or ABACUS [21] for arbitrarily approximating circuits or combining multiple approximation techniques in a single application. Most of these single-purpose methods have been demonstrated well using different applications, and the general-purpose methods combining several single-purpose methods could yield higher benefits in real-world applications. However, combining multiple approximations in an application exposes various design challenges. When multiple approximation techniques are used in conjunction, the design space size grows exponentially with the number of employed techniques due to distinct parameters exposed by each technique. In addition, the propagation of approximation error through different system components and their collective influence on the output quality need to be controlled in order to guarantee acceptable application quality. Due to the conflicting resource and quality objectives in approximate computing, an effective multi-objective DSE, guided by an appropriate optimization algorithm, is necessary where an exhaustive search becomes infeasible due to the high complexity of the design space. To ensure timely evaluation of the solutions, the estimation of quality and resources, such as power consumption, area and performance, should be fast and avoid the time-consuming synthesis of the system for each parametrization. Furthermore, for a designer to make valid choices regarding acceptable quality degradation, the employed quality model should be interpretable and suitable for the targeted application. Addressing these challenges in FPGA-based image processing applications, *A DSE Framework for Approximate Computing Using Genetic Algorithm (AxCGA)* is proposed that can combine multiple approximations in an application and explore the resulting design space using genetic algorithm (GA) to determine the quality-resource trade-off for suitable design decisions.

1.2 Scope of the Thesis

This thesis work is conducted as part of a collaborative project, *ACIP - approximate computing for professional image processing*, which aims to create resource-efficient FPGA-based approximate image processing applications [22]. In this project, the AxCGA framework is jointly developed with the project partners to combine multiple approximation techniques and achieve resource benefits in image processing applications. The overall work involved in developing the framework can be broadly categorized into: 1) a modeling part responsible for providing resources and quality values and 2) a DSE part responsible for determining the quality-resource trade-off.

The core contribution of this thesis is the DSE methodology used in AxCGA which efficiently explores complex design spaces, enabling an application designer to assess the quality-resource trade-off to make accurate design decisions in an application. A data flow graph (DFG)-based approach is adopted in AxCGA to define an application where each node represents a specific operation or component which can be replaced by an approximate component or a function from a pre-characterized library of approximate components. The proposed AxCGA framework configures and optimizes exposed parameters from all the approximations techniques in conjunction using GA and Non-dominated Sorting Genetic Algorithm-II (NSGA-II)-based selection. The parameters from different components are encoded by considering the interdependencies, and the GA-based optimization is performed on the joined parameter set in the AxCGA. Due to this global optimization on the combined parameter set, interactions and error propagation between different system components are implicitly considered in AxCGA. This thesis additionally contributes a novel *Region of Interest Non-dominated Sorting Genetic Algorithm-II (ROI-NSGA-II)* approach, which replaces the classical NSGA-II and helps AxCGA to concentrate the search pressure into specific regions in a design space. These regions are derived from the additional quality-threshold parameter introduced in approximate computing, and ROI-NSGA-II efficiently determines the quality-resource trade-off within this region where a designer is particularly interested. Additionally, a novel approach is proposed to provide hyperparameters adaptively during the DSE, which avoids the time-consuming hyperparameter optimization for each application. To further increase the efficiency, AxCGA framework uses fast models to estimate application quality and resource usage without the need for time-consuming synthesis during optimization. These models and the library of the approximate components are mainly developed by Simon Conrady and Arne Kreddig as a part of the collaboration and are not contributions of this thesis. However, a brief description of these models and the approximate computing library is also included in this thesis for a better understanding of the AxCGA framework.

Currently, AxCGA framework is limited for FPGA-based image processing applications that stream pixels without any feedback loops since the models are developed explicitly for the FPGA platform. However, using suitable approximate components and resource quality models, the proposed DSE part can easily support other application domains and target platforms.

1.3 Research Questions

This thesis primarily aims to address the complex DSE problem to determine the quality-resource trade-off in FPGA-based image processing applications when multiple approximation techniques are combined. The following research questions are addressed to achieve this targeted goal.

- **How well do the state-of-the-art approaches perform in the context of FPGA-based approximate image processing applications?**

  Many state-of-the-art approximate computing methods exist in the literature, targeting different platforms, application scopes, or abstraction levels. Addressing this research question, this thesis investigates state-of-the-art single-purpose approximate computing techniques and general-purpose design flows which combine multiple single-purpose approximations in an application in Chapter 2. The characteristics and shortcomings of these approaches are discussed in the context of approximate image processing on FPGA. Additionally, performance comparisons between one of the prominent state-of-the-art methods, namely autoAx, and proposed AxCGA variants are included in Chapter 3 and 4.

- **How to effectively explore a design space exposed by multiple approximations and determine the quality-resource trade-off in an FPGA-based application?**

  Combining multiple approximate computing methods in an application increases the design space size exponentially due to various parameters exposed by each approximation. Due to the complexity of design space and lack of an effective DSE approach, a comprehensive quality-resource trade-off analysis is often missing in state-of-the-art approaches that combine multiple approximation techniques. Therefore, possible benefits from the combined approximations are usually not well exploited. Hence, the second research question aims to address the problem of effective exploration of complex design spaces exposed by combined approximations for quality-resource analysis. Due to the structural cross-dependencies between different system components in an application, the design space exploration methods should globally optimize the parameters exposed from combined approximation methods to accommodate forward or backward dependencies of subsequent parameters. The proposed AxCGA framework based on NSGA-II selection addresses these problems in Chapter 3, and multiple experiments are performed to demonstrate the usability and effectiveness of AxCGA in real-world FPGA-based image processing applications.

- **How to incorporate the *quality-threshold* into the DSE for an efficient exploration?**

  Approximate computing introduces an additional *quality-threshold* parameter in applications, and a region of interest (ROI) in the design space can be derived based on this threshold value, where a designer can select configurations for practical implementation. Therefore, integrating this parameter into the DSE is necessary to concentrate exploration efforts into a desired region and improve computational efficiency. Addressing this research question in Chapter 4, a novel ROI-NSGA-II is proposed and integrated to AxCGA for an effective and efficient DSE in real-world approximate image processing applications.

- **How well do the ROI-NSGA-II scalable to problems with different Pareto-optimal shapes and dimensions?**

  Real-world approximate computing DSE problems often have complex Pareto-optimal shapes and multiple exploration targets. Therefore, it is essential to demonstrate the scalability of novel ROI-NSGA-II in solving problems with different Pareto-optimal shapes and dimensions to prove its generality. Addressing this problem, verification and validation of ROI-NSGA-II is performed on a set of optimization benchmark problems in Chapter 5.

- **How to avoid time-consuming application dependent hyperparameter tuning in AxCGA?**

  The proposed AxCGA exposes different hyperparameters to explore a design space effectively, and tuning these parameters for each target application is either very time-consuming or prohibitively long. Therefore, a suitable strategy is necessary to supply these parameters adaptively during the optimization. In Chapter 3, a novel adaptive GA approach is presented, which dynamically adjusts the hyperparameters during optimization.

1.4 Contributions

This thesis contributes concepts and methods for efficient DSE in approximate image processing on FPGA. The contributions of this thesis can be summarized as:

- The first contribution of this thesis is a comprehensive survey of single-purpose approximate computing approaches and general-purpose approximate computing design flows that combine multiple approximations in an application. Each general-purpose design flow is analyzed and briefly discussed its shortcomings in addressing the problem of combining multiple approximation techniques in an application.

- Overcoming limitations of may state-of-the-art approaches, secondly, this thesis contributes a GA-based approach for an effective DSE in AxCGA framework. Additionally, the AxCGA approach is demonstrated on two FPGA-based image processing applications that combine multiple approximation techniques, and the AxCGA DSE performance is compared with a state-of-the-art autoAx DSE. The genetic operations are proposed to efficiently explore the design spaces introduced in these applications.

- The most important contribution of this thesis is a novel ROI-NSGA-II optimization approach that integrates the user preference into a DSE. The ROI-NSGA-II is integrated to AxCGA for an efficient and effective DSE and demonstrated with the same applications mentioned above. Additionally, the ROI-NSGA-II is verified and validated on a broader set of optimization benchmark problems to demonstrate the scalability and compare its performance with a relevant state-of-the-art approach.

- Another important contribution in this thesis is a novel adaptive approach that can dynamically supply hyperparameters during the DSE. This approach avoids the need for time-consuming and application-specific hyperparameter optimizations in AxCGA.

1.5 Publications

Multiple parts of this dissertation have been previously published in the following peer-reviewed journals, conferences, and workshops.

[23] M. Manuel, A. Kreddig, S. Conrady, N. Anh Vu Doan and W. Stechele. Model-Based Design Space Exploration for Approximate Image Processing on FPGA. In *2020 IEEE Nordic Circuits and Systems Conference (NorCAS)*, pages 1–7, October 2020. DOI: `10.1109/NorCAS51424.2020.9265138`.

[24] M. Manuel, A. Kreddig, S. Conrady, N. A. V. Doan and W. Stechele. Region of Interest-Based Parameter Optimization for Approximate Image Processing on FPGAs. *International Journal of Networking and Computing*, 11(2):438–462, 2021. DOI: `10.15803/ijnc.11.2_438`.

[25] M. Manuel, B. Hien, S. Conrady, A. Kreddig, N. A. V. Doan and W. Stechele. Region of interest based non-dominated sorting genetic algorithm-II: an invite and conquer approach. In *Proceedings of the Genetic and Evolutionary Computation Conference*, GECCO ’22, pages 556–564, New York, NY, USA. Association for Computing Machinery, July 2022. ISBN: 978-1-4503-9237-2. DOI: `10.1145/3512290.3528872`.

[26] S. Conrady, M. Manuel, A. Kreddig and W. Stechele. LCS-based automatic configuration of approximate computing parameters for FPGA system designs. In *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, GECCO ’19, pages 1271–1279, New York, NY, USA. Association for Computing Machinery, July 2019. ISBN: 978-1-4503-6748-6. DOI: `10.1145/3319619.3326820`.

[27] N. A. Vu Doan, M. Manuel, S. Conrady, A. Kreddig and W. Stechele. Parameter Optimization of Approximate Image Processing Algorithms in FPGAs. In *2020 Eighth International Symposium on Computing and Networking Workshops (CANDARW)*, pages 74–80, November 2020. DOI: `10.1109/CANDARW51189.2020.00026`.

[28] A. Kreddig, S. Conrady, M. Manuel and W. Stechele. A Framework for Hardware-Accelerated Design Space Exploration for Approximate Computing on FPGA. In *2021 24th Euromicro Conference on Digital System Design (DSD)*, pages 1–8, September 2021. DOI: `10.1109/DSD53832.2021.00010`.

[29] S. Conrady, A. Kreddig, M. Manuel, N. A. V. Doan and W. Stechele. Model-Based Design Space Exploration for FPGA-based Image Processing Applications Employing Parameterizable Approximations. *Microprocessors and Microsystems*, 87:104386, November 2021. ISSN: 0141-9331. DOI: `10.1016/j.micpro.2021.104386`.

1.6 Outline

The proposed AxCGA framework has been developed over the past years and was presented at distinct stages in several publications [23, 24, 25, 26, 27, 28, 29]. Therefore, the following chapters in this thesis are inherited from these publications. However, this thesis uses the name AxCGA for the proposed framework for the first time, includes an updated version of the DSE methodologies, and presents the improved results from these publications. In general, the remaining chapters in this thesis are organized as follows.

Chapter 2: Chapter 2 gives an overview of various state-of-the-art approximate computing methods targeting different application scopes and abstraction levels. These approaches are classified broadly into single-purpose and general-purpose methods based on the scope, where the single-purpose methods often target a specific system component or functionality, and the general-purposed methods often combine multiple single-purpose methods in an application. Finally, the characteristics of the state-of-the-art general-purpose methods and the necessity of a renewed approach are discussed in detail for approximations in FPGA-based applications.

Chapter 3: The proposed AxCGA framework, which uses NSGA-II to explore complex design space and determine quality-resource trade-off, is explained in Chapter 3. This chapter introduces two different image processing case studies and demonstrates the application of AxCGA in these two case studies. This chapter also describes the approximate computing techniques used in these case studies, the corresponding design space formed from their parameters, and the DSE results from multiple experiments. Additionally, a performance comparison between proposed AxCGA and autoAx DSE approach and a comparison between adaptive and non-adaptive AxCGA approaches are also performed in this chapter.

Chapter 4: In Chapter 4, the ROI-NSGA-II AxCGA, which incorporates the quality-threshold parameter to the search, is presented. In addition, the three-phase implementation of the ROI-NSGA-II approach is described in detail. Thereafter, the ROI-NSGA-II AxCGA experimental results on the case studies are compared with both the NSGA-II AxCGA and autoAx DSE. Finally, a comparison between adaptive and non-adaptive ROI-NSGA-II AxCGA is also performed in this chapter.

Chapter 5: In Chapter 5, verification and validation of the novel ROI-NSGA-II are performed on different test problems to demonstrate its scalability. A set of ZDT and DTLZ optimization benchmark problems with two to ten objectives and different shapes and structures of the Pareto-optimal front are considered. Additionally, the optimization results of ROI-NSGA-II are compared with a relevant state-of-the-art approach.

Chapter 6: Finally, Chapter 6 concludes this thesis briefly by summarizing the AxCGA approach and highlights the advantages of using AxCGA for approximate image processing on FPGA. In addition, this section summarizes the limitations of the current version of AxCGA and gives a short outlook on future work to overcome these limitations.

CHAPTER 2

Related Work

Modern computing systems face numerous challenges due to the overwhelming demands for computing resources. The research fields like approximate computing aim to address these challenges by introducing new computing measures, gaining wider research attention in recent years. Due to the rising interest from researchers, multiple surveys on approximate computing techniques have already been published [9, 13, 14, 15, 17, 18, 19], and these surveys provide a broader overview of the main research categories and numerous techniques proposed before. The existing surveys use different taxonomies to classify the approximate computing techniques. Overall, state-of-the-art approximate computing techniques can be broadly classified into five categories based on the *approximation scopes, abstraction levels, error types, target computing platforms, and approximation objectives*, as shown in Figure 2.1.

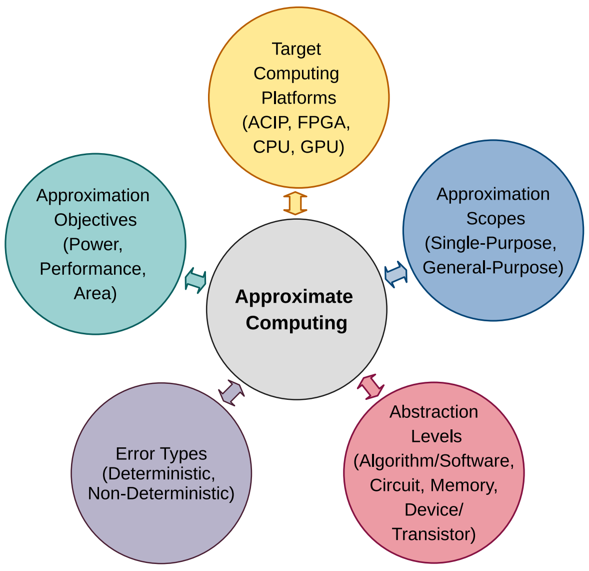

Target Computing Platforms (ACIP, FPGA, CPU, GPU)
Approximation Objectives (Power, Performance, Area)
Approximation Scopes (Single-Purpose, General-Purpose)
Approximate Computing
Error Types (Deterministic, Non-Deterministic)
Abstraction Levels (Algorithm/Software, Circuit, Memory, Device/ Transistor)

Figure 2.1: Approximate computing taxonomies in literature

The first *approximation scopes* classifies the state-of-the-art techniques into *single-purpose* or *general-purpose* approaches. Approximate computing methods that use a specific technique, such as precision scaling or voltage overscaling, or target a particular system component, such as an adder or multiplier, are known as single-purpose approximation methods. On the other hand, general-purpose approaches have been proposed to approximate arbitrary circuits, approximate multiple system components, or employ one or more single-purpose techniques simultaneously in an application. Additionally, the general-purpose methods that approximate arbitrary circuits can potentially be applied to produce a single-purpose approximation technique such as an adder, multiplier, or divider as well. A search heuristic is often associated with these methods to identify a suitable combination of involved approximation methods and their tunable parameters. The second *abstraction levels* categories the state-of-the-art approximate computing techniques based on the level of abstraction where a specific technique is targeting, such as algorithm or software level, memory level, circuit level, and device or transistor level. Since different types of approximation techniques are often involved in general-purpose methods, some approaches can simultaneously consider approximation techniques in multiple levels of abstraction [21, 30, 31].

The third *error types* classification is based on the reproducibility of errors in an approximation technique for the given specific inputs. The deterministic approaches always exhibit the same output behavior for the same set of inputs, whereas the non-deterministic approaches might behave differently. Due to this limited reproducibility, applications with non-deterministic techniques are challenging in testing and debugging. Therefore, a designer must carefully consider the quality degradation in such techniques during the design phase to meet the quality-threshold. On the other hand, since the error behavior is predictable in deterministic methods, a designer can directly specify a quality-threshold in an application and consistently evaluates whether the threshold is met for the desired set of inputs. The fourth *target computing platforms* classification is based on the type of platform an approximation technique can be applied, such as FPGA, ASIC, graphics processing unit (GPU) or CPU. Finally, the state-of-the-art techniques can further be classified based on the desired *approximation objectives* such as performance, area, and power or energy at the expense of application quality.

This chapter provides a comprehensive survey of promising state-of-the-art works in approximate computing research and classifies them based on the scope of approximation and targeted level of abstraction, as shown in Figure 2.2. This chapter is further structured into two main sections based on the scope to describe each technique briefly. The first Section 2.1 investigates state-of-the-art single-purpose approximate computing techniques, which are demonstrated well in isolation, whereas the second Section 2.2 reviews state-of-the-art general-purpose design flows that consider multiple techniques simultaneously in an application.

2.1 Single-Purpose Approximate Computing Methods

Over the last decades, various single-purpose approximation techniques have been proposed to exploit the error resilience of applications. These approaches can be further classified based on the other taxonomy dimensions, such as abstraction levels, error types, target computing platforms or approximation objectives. However, the boundaries within each taxonomy dimension often overlap each other, especially within target computing platforms or approximation objectives. For example, circuit-level techniques optimized for a reduced number of gates in an ASIC design can

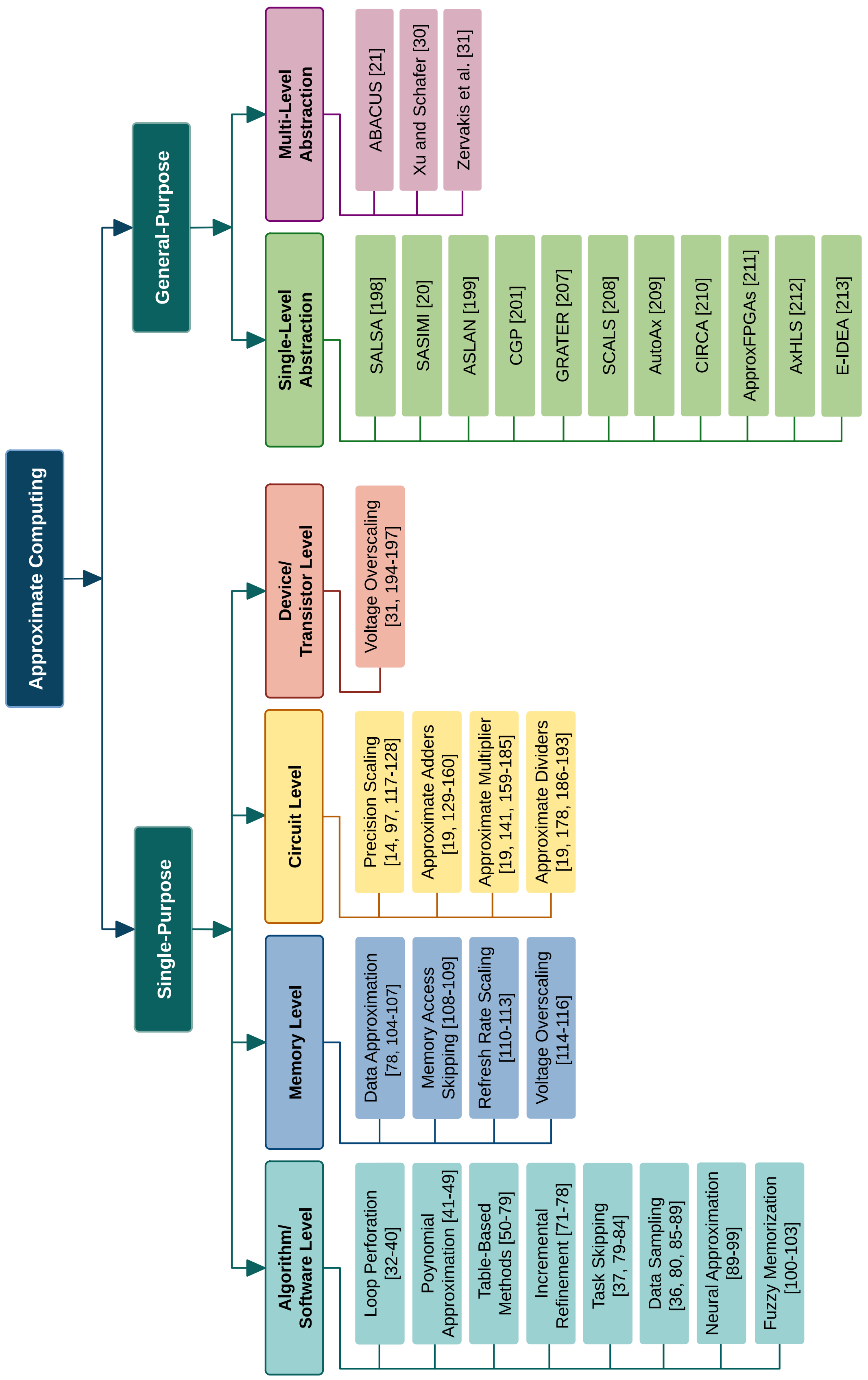

Approximate Computing
Single-Purpose
General-Purpose
Algorithm/ Software Level
Memory Level
Circuit Level
Device/ Transistor Level
Single-Level Abstraction
Multi-Level Abstraction
Loop Perforation [32-40]
Poynomial Approximation [41-49]
Table-Based Methods [50-79]
Incremental Refinement [71-78]
Task Skipping [37, 79-84]
Data Sampling [36, 80, 85-89]
Neural Approximation [89-99]
Fuzzy Memorization [100-103]
Data Approximation [78, 104-107]
Memory Access Skipping [108-109]
Refresh Rate Scaling [110-113]
Voltage Overscaling [114-116]
Precision Scaling [14, 97, 117-128]
Approximate Adders [19, 129-160]
Approximate Multiplier [19, 141, 159-185]
Approximate Dividers [19, 178, 186-193]
Voltage Overscaling [31, 194-197]
SALSA [198]
SASIMI [20]
ASLAN [199]
CGP [201]
GRATER [207]
SCALS [208]
AutoAx [209]
CIRCA [210]
ApproxFPGAs [211]
AxHLS [212]
E-IDEA [213]
ABACUS [21]
Xu and Schafer [30]
Zervakis et al. [31]

Figure 2.2: Classification of approximate computing literature based on the scope of application and targeting abstraction levels

also reduce the resource utilization in FPGA. Similarly, approximation techniques that reduce the area in a design often reduce power consumption as well. Additionally, the majority of approximate computing techniques in the literature are deterministic in nature since the quality-threshold bounds in an application have to be met importantly. Therefore, this thesis further classifies the single-purpose approximate computing techniques based on the targeting abstraction level in an application. The following sections briefly describe each level of abstraction and the approximate computing techniques that can be applied to these levels.

2.1.1 Algorithm or Software Level Approximations

Approximate computing at the algorithm level often utilizes statistical properties of data, such as data sampling, or parts of an algorithm being implemented at a coarser granularity, such as loop iterations, to trade in quality for hardware resources. These methods often benefit resources such as performance, power, or area, irrespective of the target computing platforms. For example, implementing an algorithm by skipping some tasks or iterations of a loop might potentially benefit in terms of performance or reduced power consumption across various target platforms, including in the FPGA applications. Since many of these approaches approximate a specific part of the algorithm or functionality that does not significantly affect the final application quality, these methods are sometimes categorized as software-level functional approximation techniques in the literature. Different types of algorithm or software level approximation techniques are briefly described in the forthcoming subsections.

2.1.1.1 Loop Perforation

One of the most commonly used algorithm level approximation techniques, loop perforation [32, 33, 34, 35, 36, 37, 38, 39, 40], selectively skips iterations in a loop. Most of the loop perforation methods determine approximable loop candidates using certain preliminary mechanisms that analyze the instructions or operations in an application and then transform these loops to skip certain iterations in a controlled manner. However, despite the traditional approaches that skip iterations in a fixed manner, dynamic perforation methods can skip iterations adaptively at different computations phases. Li et al. developed Sculptor tool, which uses both selective perforation, where the loops are transformed to skip a subset of their iterations during program execution, and dynamic perforation, where loops are transformed to skip a flexible subset of their iterations using a built-in scheduler [39]. The Sculptor trades these selective and dynamic perforations techniques in runtime for error management, and the system adjusts the skip rate based on the error difference between the measured and target error. Similarly, Mitra et al. [40] presented OPPROX, which identifies different computation phases in a program and performs a phase-specific trade-off space exploration to determine the most profitable perforation settings at each phase.

2.1.1.2 Polynomial Approximation

Elementary mathematical functions and transcendental functions are ubiquitous, including in images or signal processing applications. However, accurately evaluating such functions is often computationally expensive on a system, both at the software and hardware level. Therefore, these computations are frequently replaced with polynomial functions of a specific degree, and such approximations can bring considerable benefits to computations. In 1996, Jean-Michel Muller presented various polynomial approximation techniques for computing elementary functions [41]. This proposal includes both least-squares approximations, where the approximations minimize the average error and least maximum polynomial, which targets to minimize the worst-case error in computations. The polynomial functions can be implemented on hardware with a number of multiplications and additions, and the multiplier size is often a significant concern in terms of hardware resources. A straightforward way to improve the resource utilization is to lower the degree of approximating polynomial in a controlled manner. The benefits of changing the polynomial order in FPGA resource utilization have been demonstrated by Nagayama et al. [42]. Additionally, forcing the polynomial coefficients to fit into a small number of bits also reduces the further multiplication efforts [43]. Brisebarre et al. used sparse coefficients to achieve small-size multiplications in polynomial approximations [44]. Chevillard et al. presented Sollya, a complete toolchain that can generate fixed- and floating-point software and hardware design for mathematical functions with approximations [45]. Similarly, FloPoCo, proposed by Dinechin [46], is an automated code generator for FPGAs that tries to minimize the size of the coefficient and, thereby, multiplier size in polynomial approximation. The resulting architecture efficiently uses the resource on FPGAs. Later, Lauter proposed Metalibm, which divides the calculation interval into several sub-intervals and then employs Sollya in each interval to obtain an approximate polynomial that meets the target accuracy [47]. However, in very large intervals, Metalibm faces difficulty in detecting the properties of target transcendental functions, resulting in reduced performance. Overcoming this issue, TGen proposed by Hao et al. [48], an automatic generator of a variable precision transcendental function, first converts the target function to a combination of basic functions and then uses Sollya to complete the polynomial approximation in the sub-interval [48]. Decades later, in 2020, Muller performed a review of approximate computing techniques for the polynomial approximations and concluded that continuous arbitrary functions could be approximated within the desired accuracy using polynomial approximations [49]. However, the freedom of approximation using simple coefficients is tailored to the implementing function and accuracy target within the target bounds of the functions.

2.1.1.3 Table-Based Approximation

Table-based approximate computing techniques are generally used for implementing complex continuous functions. Many methods have already been proposed, ranging from simple approaches that store values of a function for all the inputs to more sophisticated approaches that keep a few values in a table and interpolate the intermediates using arithmetic operations.

In general, based on the size of the table and computation complexity, the table-based methods are further classified into three categories such as compute-bound methods, table-bound methods, and in-between methods. First, compute-bound methods primarily use arithmetic computations to evaluate a function with a small table lookup to store essential parameters. A bunch of methods proposed by Ping-Tak Peter Tang [50, 51, 52, 53] are typical examples of compute-bound and are used widely in general-purpose computing systems. Secondly, the table-bound methods store large tables and use very few computations, such as simple additions, to reconstruct the function value. Bipartite table methods are typical examples of this class and consist of two tables and an addition [54]. The symmetric bipartite table [55] and multipartite table methods with more than two tables [56, 57] are also further modifications of bipartite table methods. Since the table size increases exponentially with an increase in accuracy, these methods are suitable for low-precision applications. Lastly, in-between methods use a medium-sized table lookup with a significant yet reduced computation. The in-between methods are popular for a better trade-off between accuracy and resources for middle-to-high-precision applications. These methods are further categorized into linear and polynomial based on the type of employed approximations.

A commonly used in-between approximation method is a piecewise approximation technique. The input of a target function is divided into several segments, and each segment is approximated using linear or polynomial functions. The piecewise linear (PWL) approximation techniques generally require storing only the starting point of each segment, slope, and y-intercept in a table [58]. In many piecewise polynomial (PWP) approximations, the input is split into two parts: a most significant part X_1 and a least significant part X_2. The target function is uniformly divided into the power of 2 (2^{X_1}) segments. Then each segment is approximated using a quadratic polynomial $aX_2^2 + bX_2 + c$. The coefficients a, b, and c can be indexed directly using X_1 on the hardware level [59]. Hsiao et al. proposed two-level approximations to reduce the total area and delay [60]. A piecewise degree-one polynomial is used in the first level of approximation and another piecewise quadratic interpolation or table lookup is used in the second phase of refined approximations.

Depending on the type of segmentation, the piecewise approximation can be further divided into uniform segment based, non-uniform segment based, and error-flattened on how the input ranges are divided. The uniform segment based approximations divide the input range into equal-length segments, and then a linear or polynomial function approximates each uniform segment [61, 62]. However, the slope or variation rates of nonlinear target functions are different in distinct input regions. Therefore, these methods need more segments in regions with significant variations, and this might lead to the use of additional computing resources than they actually require in low-variation regions. The non-uniform piecewise approximations are proposed for overcoming these drawbacks, and Nam et al. [63, 64] used 15 segments to compute a logarithmic function. Even though a relatively better accuracy is achieved with fewer resources, the error ranges are still uncontrollable in certain regions. This category includes multi-level hierarchical segmentation, in which each segment can be further divided into sub-segments [65, 66, 67]. Such techniques have finer sub-segments in high-variation regions and only a few sub-segments in low-variation regions, thereby reducing overall resource utilization. The error-flattened piecewise approximations guide the division of segments in a way that each segment satisfies error bounds and guarantees the same error rates in each region. Zhu et al. demonstrated an approach that uses maximum relative error to convert logarithmic functions [68]. Lie et al. proposed an input segmentation method based on the maximum absolute error on corresponding equally divided output segments, which is more suitable for hardware efficiency [69]. A few other methods by Sun et al. [58] Dong et al. [70] generalize the piecewise approximations by decoupling the characteristics of target functions from the segmentation scheme.

2.1.1.4 Incremental Refinement

Incremental refinement approximate computing approaches often consist of a succession of computation stages and improve the accuracy over these stages. The name incremental refinement is often synonymously used as iterative or successive refinement in the literature. A typical example of incremental refinement is Newton's root-finding method, in which a required accuracy determines the number of iterations has to be performed. Winograd and Nawab initially presented a class of multistage algorithms that use incremental refinement for signal processing applications such as discrete Fourier transform (DFT) and short-time Fourier transform (STFT) approximations [71]. The same research group later demonstrated how the existing recursive structures could be adapted as incremental refinement approaches on applications such as signal detection using fast Fourier transform (FFT), spectral analysis using DFT, discrete cosine transform (DCT)-based image encoding or decoding, and digital filters [72, 73]. Andreopoulos and Patras employed incremental refinement for the salient-point detection with reduced computational complexity and energy consumption [74]. Several iterative refinement algorithms are published with reduced precision in intermediate computations [75, 76, 77, 78], and a same quality of results can be achieved with additional iterations. However, early termination in the iterative refinement can further improve computation performance.

2.1.1.5 Task Skipping

Task skipping is also a popular algorithm level approximate computing technique for trading the accuracy for computational resources. However, it requires a careful selection of approximable parts to skip some computations and keep the induced error in a controlled manner. The loop perforations in the previous section that skip particular iterations are also a task skipping approach. Martin Rinard proposed an approach for task skipping that allows a designer to identify the quality distortions that fall within the acceptable bounds using probabilistic models [79]. This approach characterizes the quality distortion and execution time as a function of the task failure rates, enabling the accuracy-execution time trade-off. Similarly, Goiri et al. presented a framework ApproxHadoop for creating and running approximations such as task skipping and estimating the corresponding error bound in MapReduce programs [80]. The ApproxHadoop is demonstrated in a class of MapReduce programs and could significantly reduce the execution time and energy consumption. Byna et al. showed that dropping non-critical tasks in supervised semantic indexing algorithms improves the performance in GPU [81]. However, this approach exploits the unique characteristics of an application and is, therefore, an application-specific approach. SAGE proposed by Samadi also targets GPUs by automatically detecting and systematically skipping computationally expensive operations, and this approach trade-offs the computation accuracy for the performance [37]. Several other approaches are also successfully demonstrated the task skipping approaches in different application domains [82, 83, 84].

2.1.1.6 Data Sampling

Data sampling approximate computing techniques utilize a subset of actual data to be processed by a computational system and trade the computational accuracy for system performance. Data sampling techniques are used in many applications, such as machine learning or deep learning, big data, or database management systems. Approximate queries from databases widely use data sampling-based techniques to leverage the accuracy for the query processing time [85, 86, 87, 88]. Ansel et al. proposed a novel programming model incorporating variable accuracy choices to the algorithms where the approach automatically searches and finds optimal algorithmic closes for each target accuracy level [89]. This approach includes varying input sizes for the accuracy-performance trade-off and demonstrates the applicability on six benchmark problems. Similarly, Paraprox, proposed by Samadi et al. applied data-sampling on data-parallel programs that contain common computation patterns [36]. Goiri et al. integrated an approximation mechanism that uses data sampling together with task skipping to the MapReduce paradigm and demonstrated the approach in applications from different domains, including data analytics, scientific computing, video encoding, and machine learning [80].

2.1.1.7 Neural Approximation

Neural approximation is a general-purpose function approximation method that approximates an unknown underlying function from available observations in an application. The research about using artificial neural network (ANN) in the functional approximation started many decades ago. In 1987, Wieland and Leighton [90], and in 1988, Irie and Miyake [91] studied the capabilities of multilayer perception in approximating the functions. Later, multiple pieces of research were published addressing the neural approximation techniques for approximating arbitrary functions [92, 93, 94, 95]. In 2012, Esmaeilzadeh et al. proposed a neural approximation approach that accelerates general-purpose programs with the support of a reconfigurable digital neural processing unit (NPU) [96]. Their approach uses a Parrot transformation technique that converts a region of code the designer annotates into a common neural intermediate representation. Due to the high-level parallelism of the neural networks, the NPU accelerates the execution of code energy efficiently. This work considers a set of error-resilient benchmark applications, and the NPUs achieved an average 2.3x speedup and 3.0x energy saving on these applications. A similar algorithmic transformation approach by Amant et al. automatically converts approximable code regions to a limited precision analog neural representation for accelerating on analoge neural processing unit (A-NPU) [97]. Their analog acceleration technique could achieve an average of 3.7x speedup and 6.3x energy saving on the benchmark applications. McAfee and Olukotun presented a neural network based platform for emulation and acceleration of applications called EMEURO [98]. Their approach emulates an application by breaking it into several subtasks and determining the suitability of approximation for each subtask using an approximate programming model proposed by Ansel et al. [89]. Afterward, a large set of hybrid task graphs, including a mixture of exact and approximate subtasks with various errors and latency for different input characteristics, is determined during the compilation process. This phase uses a large set of simple 2-layer linear neural network to approximate these subtasks. Finally, the EMEURO efficiently selects one of the subtasks that meet the design preference during the runtime. The EMEURO system is demonstrated using different applications and could achieve a significant 109x speedup with a bounded error between 0.1% to 10%. Eldridge et al. employed neural network based approximation for floating-point transcendental functions [99]. Their approach trains a model for approximating a function in a limited range, and they compute the function value for any interval using this model.

2.1.1.8 Fuzzy Memorization

Fuzzy memorization is an approximation technique that combines instruction memorization [100], where the operands and results of certain operations such as multiplication or division are stored in a reusable table with fuzzy computation [101], which tolerates inexactness in computation results to reduce the latency and saves power consumption. Álvarez et al. proposed a fuzzy memorization technique for floating-point multimedia applications [102]. Their approach uses a conventional floating-point unit and a memory table to store a fixed number of input operands, operations, and the results of floating-point operations. When executing a floating-point operation, it searches first in the table. However, n *LSB* bits are removed from mantissa before searching the table. Therefore, a range of similar inputs hits the entries in the table without executing the floating-point operation. The tolerance level and table sizes can vary with different n, and different quality-resource trade-offs can be obtained. To improve the output quality of fuzzy memorization, Ono and Usami proposed an approach that adds a value calculated using a simplified formula to the table result to produce the final output in case of a table hit [103]. The difference between the actual input operands and the input operands in the table is used for the simplified formula. This approach is demonstrated on an FPGA using a grayscale conversion application and could improve the performance and energy savings at the expense of quality.

2.1.2 Memory Level Approximations

Memory units are critical components in a computational system to store instructions or data, and the memory types vary depending on the technology used in storage. The modern FPGA-based systems comprise multiple memories such as logic blocks or logic elements inside the FPGA fabric, which configure for implementing logic functionalities, and block RAMs (BRAMs), which use for storing large amounts of data within FPGA, and often includes other external memory units such dynamic random-access memory (DRAM) or static random access memory (SRAM). The memory costs such as size, performance, or energy consumption are often bottlenecks in many data-intensive applications, especially in the edge or mobile applications. Therefore, these components are strong candidates for approximate computing, and different memory level techniques have been proposed that exploit the application quality for the resource benefits.

2.1.2.1 Memory Data Approximation

The bandwidth or energy consumption throttles the capabilities of memory systems in many applications. Addressing this issue, memory data approximate mechanisms approximate the data that is stored, loaded from, or written to the memory for relaxed application quality. Either the typical memory operations or the storage mechanisms are altered in the literature to achieve approximate memory data. Fang et al. proposed an approach for reducing energy-expensive write operations of data to a phase change memory in video applications [104]. Before writing new data, it first reads the old data in the exact location and computes the absolute difference between them. This approach cancels the write operation if the difference is below a certain threshold. Ranjan et al. proposed approximate storage for energy-efficient spintronic memories by employing specific read or write operations [105]. After the characterization of the energy-quality trade-off, a quality-configurable memory array is used to perform read or write operations for approximating data. Depending on the application requirements, various predefined accuracy levels can be chosen in runtime. Targeting the solid-state memories, Sampson et al. presented two different approximate storage mechanisms in multi-level cell memory to improve the energy efficiency, performance, lifetime, and capacity of the memory component [106]. The first mechanism uses a reduced number of programming pulses to write data at the cost of occasional inaccuracies, and the second mechanism uses worn-out or faulty memory cells from the development cycle to store approximate data together with an error correction mechanism based on bit position significance. To obtain similar benefits, Ganapathy et al. used the faulty cells to store the lower significant bits using a hardware mechanism that circular shifts the data [107]. Therefore, the error distribution is skewed towards the lower-order bit. Tian et al. presented a framework, ApproxMA, to identify the required precision of data to be accessed from the external memory using a runtime precision controller and load the scaled data for computation using a memory access controller [78]. This scaled memory access and the further computations with reduced precision decrease the overall memory resource utilization.

2.1.2.2 Memory Access Skipping

Memory access often significantly consumes energy, and Zhang et al. proposed ApproxANN, which incorporates memory access skipping in ANN applications [108]. Their approach initially performs a criticality analysis of each neuron by considering the influence of injected error on the output. Thereafter, skip several uncritical neurons by ignoring the reading of specific rows in a weight matrix to improve the overall energy savings. A set of ANN applications are considered for the experiments and achieved 34.11% to 51.72% energy benefit with less than 5% quality loss. Similarly, Qin and Idreos proposed an idea for adaptive data skipping from main memory for various applications, and their initial results showed 1.4x performance improvement [109].

2.1.2.3 Refresh Rate Scaling

The technologies used to store data in memories often lead to certain technology-dependent constraints. DRAMs are commonly used main memories for storing very large amounts of data in computational systems, and an FPGA based system usually associates with an external DRAM module. The DRAM technology requires memory refresh to periodically rewrites the data to the memory cells, thereby often leading to high energy consumption. Approximations can be incorporated in this energy-expensive refresh rate mechanism at the expense of data error in error-resilient of applications.

Liu et al. introduced Flikker to reduce DRAM refresh power [110]. This approach allows a designer to specify the critical or non-critical data in programs, and the data is assigned to different memory locations. The non-critical data is then refreshed at a lower rate to save energy by keeping the refresh rate of the critical data at regular intervals. Therefore, the corresponding error in the non-critical data is often tolerated by the error resilience of applications. The Flikker approach is experimented with different applications and saved DRAM memory power up to 25% with less than 1% performance degradation and no loss in application reliability. Chao et al. proposed Tiered Reliability Memory (TRM) approach to save power in embedded DRAM buffers in video applications [111]. Similar to the Flikker approach, the DRAM buffers are divided into multiple segments with different refresh rates. However, the data is stored in each segment on sub-pixel granularity such that the most significant bits (MSBs) are stored on the most reliable segment whereas least significant bits (LSBs) are stored on the least reliable segment. The experiments showed that the TRM saves 48% refresh power without any significant loss of visual quality. Ganapathy et al. relaxed the constraints that allow the embedded DRAM refresh rate to move beyond the worst-case point of failure and thereby improved the memory availability and energy efficiency in data-intensive error-resilient applications with a slight loss in application quality [112]. Quality configurable approximate DRAMs are proposed by Raha et al. based on extensive error characterization using refresh rate as a quality control knob [113]. Improving the Flikker approach, their technique proposed four sorting strategies for staggering data into different quality bins, which are used for systematically allocating critical and non-critical data. This approach also showed savings on DRAM power up to 73% on average from different applications with minimal error in quality.

2.1.2.4 Voltage Overscaling

Voltage overscaling is an approximate computing technique in which the supply voltage to the memory cells is overscaled to reduce power consumption. This approximation technique can be applied to both the device and the memory levels, and multiple variants have been proposed to achieve power savings. Chang et al. proposed a hybrid memory array for SRAM with a mixture of 6 transistor (6T) and 8 transistor (8T) cells to store the pixels in video applications [114]. The lower-order bits are stored in the conventional 6T cells, whereas the most significant part is stored in the robust 8T cells. Thereafter, the scaling of voltage is performed where the data in 6T cells are approximated, and 8T cells are relatively unaffected. Additionally, due to the area overhead of 8T cells, this approach optimized the number of bits that need to be stored in 8T cells, minimizing the overall approximation error. Their experiments showed that the voltage overscaling with a hybrid memory array could save at least 32% of power compared with traditional 6T only memories. Esmaeilzadeh proposed Truffle, a microarchitecture design where the components use dual-voltage SRAM cells in a way that the precise operations use high voltages domains and approximate operations use low voltage domains [115]. Their approach includes an instruction set architecture (ISA) extension for specifying the instructions that can be approximated with the extension of typical in-order and out-of-order processors. Additional dual-voltage multiplexers and voltage level shifters multiplex the signals and enable data movement between these high and low voltage domains. Experiments with a set of benchmark problems showed energy savings up to 43% with a low degradation in output quality. Salami et al. used voltage scaling to lower the voltage guardband of BRAMs to achieve energy savings in commercial FPGAs [116]. The corresponding error behavior is characterized and stored in a fault variation map during a preprocessing step, and this map is used to assign data to BRAMs based on the importance of data. The significance of their approach is demonstrated using a neural network (NN) accelerator.

2.1.3 Circuit Level Approximations

Approximate computing at the circuit level often involves a functional simplification of circuits to achieve benefits in resources at the expense of application quality. Many approaches have been proposed targeting individual arithmetic components such as adders, multipliers, or dividers. In contrast, the methods like circuit pruning, approximate synthesis, or precision scaling can be employed generally to approximate the entire circuit or different circuit components. Since these methods simplify the functionality at the circuit level, these approaches are synonymously termed circuit-level functional approximation in the literature. The following sections briefly explain different types of state-of-the-art circuit level approximation techniques.

2.1.3.1 Precision Scaling

Precision scaling reduces the amount of data to be handled by a system by changing the bitwidth of the inputs or intermediate results, thereby saving the computational efforts and the memory needed to store data. Generally, this approximation technique can be applied at the algorithm and the circuit levels. The precision can be altered effortlessly at the algorithm level by using dedicated library operations such as rounding or ceiling and implicit datatype conversion such as $int64$ to $int32$. In contrast, the precision scaling at the circuit level ignores a specific number of the least significant bits. Numerous approximate computing works reduce the precision at the circuit level to achieve resource benefits [14]. Many of these works combine precision scaling with other approximation techniques, while some use it as a stand-alone technique with different types of scaling. Dynamically scaling the precision during runtime, Yeh et al. designed a hierarchical floating-point unit where precision is incrementally reduced at runtime and compared the energy difference with threshold precision to control any instability [117]. Multiple techniques, such as converting floating-point operations to trivial ones, use of smaller floating-point unit, and reduced precision to increase the coverage of memorization lookup are used to achieve the dynamic precision scaling. Hsiao et al. proposed a hybrid precision selection framework that employs both reduced floating-point precision and variable precision fixed point states to reduce the energy consumption in mobile GPUs [118]. Similarly, Rubio-González presented Precimonious that searches the floating-point variables in a program and reduces the precision to achieve the design goals [119]. Han et al. demonstrated the use of reduced precision in distributed deep learning applications [120], where the first few epochs use 32-bit floating-point gradients, and the last set of epochs uses 8-bit floating-point to achieve significant speed-up in training with no or only a little loss of accuracy. Similarly, several neural networks or deep learning approaches or accelerators exploited the use of reduced precision to achieve better efficiency with a slight loss of accuracy [97, 121, 122, 123, 124, 125]. Custom circuit implementations on FPGAs or ASICs often use fixed point representation. Therefore, the precision of data or signal can be controlled at the bit-level, and arbitrary precision scaling can be directly applied on the global level or to specific system components. An FPGA-based multi-precision floating-point architecture presented by Zhang et al. uses three different precision and three matrix operation modes to accelerate large-scale matrix computing [126]. Similarly, Licht et al. presented a deeply pipelined design that arbitrarily scales the precision of floating-point arithmetic to accelerate fundamental operators on FPGA [127].

Targeting FPGAs, non-linear functions, such as logarithm, can be approximated with fixed-point arithmetic, which implicitly reduces the precision and saves area [128]. However, such a fixed-point implementation at the software level might increase the execution time of such applications [13].

2.1.3.2 Approximate Adders

Adders are essential building blocks in the digital image and signal processing applications. Therefore, approximations in adder circuits have high interest from researchers, and many prior works surveyed approximate adder techniques proposed over the years [19, 129]. On the circuit level, the input operand bits are successively added from LSB to MSB with a propagating carry bit to get the final result in a typical addition. Based on the bit-level granularity of adder components used for approximations, this work generally classifies the approximate adders into three categories. The first category is single-bit approximate adders, where the approximations are performed on a single-bit full adder. The second category is multiple-bit approximate adders, where the approximations techniques consider a number of bits or a segment of an adder together. Finally, generic adders approximate the entire adder circuit or consider all the adder bits.

Single-Bit Approximate Adders: Simplifying a single-bit full adder circuit, Gupta et al. proposed imprecise adders for low-power approximate computing (IMPACT) [130] that integrate three different approximation techniques to conventional mirror adder (MA) [131]. Transistors of the MA are removed judiciously by ensuring no open or short circuit occurs to exploit the quality-power trade-off. Later, their approach is extended with two additional approximate mirror adders (AMAs) [132]. Such AMA cells are then used in the LSBs of a multiple-bit adder. Similarly, three approximate XOR/XNOR-based adders (AXAs) are proposed by Yang et al. by carefully removing the transistors from the accurate design [133]. The transistor removal is also used by Nanu et al. to approximate adders using complementary pass transistor logic (CPL) [134] and Gogoi and Kumar to approximate adders using both CPL and transmission gate (TG) [135]. Almurib et al. simplified the full adder logic to introduce approximations and proposed three inexact adder (InXA) versions [136]. Prabakaran et al. presented a novel design methodology for building approximate adders denoted as DeMAS, by considering the architectural feature of FPGAs [137]. Using this generic methodology, they proposed eight different one- two-bit approximate adders for Xilinx 7-series FPGAs and demonstrated the approximation capabilities using various 16-bit approximate adders built from them. Babu and Balaji et al. used cartesian genetic programming (CGP), an evolutionary approximate circuit design method, to create one- and two-bit approximate adders [138].

Multiple-Bit Approximate Adders: One of the common types of multiple-bit approximate adder is a segment-based adder, where the actual adder is split into multiple sub-adder segments and then performs the approximation on specific segments. A large number of techniques split an n-bit adder into two segments where the accurate part (n-1 to k) and inaccurate parts (k-1 to 0) are computed separately. The Error-Tolerant Adder (ETA)-I proposed by Zhu et al. performs a normal addition in the accurate part, whereas the inaccurate part performs a special addition without propagating the carry bits by simply checking the input bits [139]. This could reduce the overall delay and the power consumption of the adder. To further improve the accuracy of addition, an enhanced ETA-II is proposed where the carry propagation path is divided into several short paths, and the carry bits are concurrently computed on these paths [139]. Gupta et al. also used a two-segment addition similar to ETA-I, where the adder components in the inaccurate part are completely removed [132]. Instead, one of the inputs is directly selected as the sum of the inaccurate part. This thesis adapted their approach and named it as lower-select adder (LSA) in the following chapters. The median adder proposed by Celia et al. approximates the lower k bits in inaccurate part to a value $2^k - 1$ by simply setting the these bits to 1 [140]. In cases where the probability distribution of inputs is known exactly, a constant value can be set instead of the median. The lower-OR adder (LOA) approach also uses a similar two-part computation, approximating the inaccurate part with a simple bitwise OR function [141]. Additionally, a carry bit to the accurate adder part is provided by simple AND operation of $k-1^{th}$ bits of inputs. Similar to LOA, sloppy adders proposed by Albicocco et al. use bitwise OR for the inaccurate part, whereas the carry input bit is provided as 0 [142]. The optimized lower part constant-OR adder (OLOCA) proposed by Dalloo et al. generalizes and optimizes the error produced using LOA and thereby outperforms LOA [143]. Targeting the FPGA design directly, other approximate adder approaches such as hardware optimization and a near-normal error distribution (HOAANED) adder [144], Fast and Error-Optimized Approximate (FAU) adder [145], a hardware efficient approximate adder proposed by Balasubramanian and Maskell [146], and a look-up table (LUT)-based approximate adder by Becher et al. [147] also use two-segment additions where the inaccurate part and carry prediction to the accurate part are effectively approximated.

Mohapatra et al. proposed a dynamic segmentation with error compensation that divides an n-bit adder into multiple independent sub-adders by bit slicing in the data path and invokes the error compensation once an overflow of carry tracking counters of each sub adders occurs [148]. Their approach reduces the critical path of adders due to the segmentation and performs additional voltage overscaling on sub-adders to save resources further. Accuracy-Configurable Approximate (ACA) adder proposed by Kahng et al. enables a runtime configuration of computation accuracy [149]. Their approach also divides the adder into multiple sub-adders, and each has a certain overlap in the considered input bits. The segmentation reduces the critical path, and the overlap improves computation accuracy. By varying this carry chain depth of sub-adders, the performance or power consumption can be traded off with accuracy. In addition, an error detection and correction mechanism further detects the error in the sub-adders and corrects them using a simple incremental circuit. Therefore, a correct result can be achieved with the same approximate implementation at the expense of multiple error-correction stages. The Gracefully-Degrading Adder (GDA) is a reconfigurable approximate adder where multiple sub-adder units with variable length are employed [150]. A hierarchical carry-in prediction logic is multiplexed with simple carry-in computation for each sub-adder component. Additionally, the carry-in prediction can be reconfigured by varying the number of prediction components used in the prediction logic. However, this approach does not have any error prediction or correction mechanism. Shafique et al. proposed a Generic Accuracy Configurable (GeAr) adder with multiple sub-adder units of equal length where the number of result bits produced per adder segment and the number of previous bits used for the carry prediction can be varied for quality-resource trade-off [151]. A configurable error correction unit also ensures accurate results when required. The quality-area optimal Low-Latency approximate adder (QuAd) approach proposed by Hanif et al. also uses sub-adders of variable lengths with arbitrary combinations of results and prediction bits [152].

Many other techniques are designed for approximating the speculation of carry bits for sub-adder segments. The speculative carry select addition (SCSA) proposed by Du et al. [153], the carry skip scheme by Kim et al. [154], high-performance low-power carry speculative adder (CSPA) by Lin et al. [155], consistent carry approximate adder (CCA) by Li and Zhou [156], exploitation of generate signal for carry speculation by Hu and Qian [157] are also used different approximation techniques to the carry speculation to save computing resources.

Generic Approximate Adders: Addressing the challenge of implementing approximate adders with varying quality-resource requirements, generic adders are proposed. Designing application-specific approximate operators (AppAxO) for FPGA-based embedded systems proposed by Ullah et al. disables the LUTs used for carry-chains of FPGAs to introduce approximations [158]. For an n-bit adder implemented with n LUTs, 2^n possible combination of LUTs can be disabled for accuracy-performance trade-off analysis. A library of approximate adders and multipliers, named EvoApprox8b, proposed by Mrazek et al. contains 430 different 8-bit approximate adders created from 13 accurate adders using CGP [159]. In CGP, a circuit is encoded into genes for evolutionary operations, and then the evolutionary operations are performed by randomly removing some connections to form different approximated versions. Later, this library was extended with more complex adders and named EvoApproxLib [160].

2.1.3.3 Approximate Multipliers

Multiplier circuits are one of the basic building blocks in digital signal processing (DSP) applications, and modern System-on-Chips (SoCs) have dedicated DSP processors for high-performance multiplication. However, using this dedicated DSPs might not always be efficient and, in many applications, consume higher power than a non DSP-based multiplier. Therefore, approximating the multiplier circuits is an important area of research in approximate computing, and multiple techniques have already been proposed. A typical combinational multiplier has three different processing phases such as partial product (PP) generation, PP accumulation, and a carry propagate addition. The PP can be obtained with simple AND operations of multiplier bits, and the accumulation of these PPs typically uses a carry-save adder array, a Wallace tree, or a Dadda tree. Different approximation techniques can be integrated into these three processing phases. Based on the approximation type and processing phase where a specific technique target, the state-of-the-art approximate multipliers can be classified into approximate PP generation, approximate PP accumulation, and functional approximation where the multiplication function is simplified. The following sections briefly summarize different methods proposed in each category.

Approximate PP Generation: Approximate PP generation integrates approximations into smaller multiplier cells that generate PP, and then reuses these cells to create larger multipliers for a higher design benefit. The underdesigned multiplier (UDM) proposed by Kulkarni et al. uses a modified Karnaugh Map (K-Map) to design a 2x2 multiplier cell [161]. This method saves an output bit by approximating the multiplication of 11_2 and 11_2 to 111_2 instead of 1001_2, thereby introducing $(1/2)^4 = 1/16$ error rate. Vasudevan et al. extended approximations into a similar block by keeping the LSB of the output bit zero, and this further reduced the area [162]. Additionally, they introduced a three-level approximation in large multipliers in a way that the least significant PP bits have the maximum degree of approximation, the middle product has a medium level of approximation, and the most significant PP has no approximation. This improves the overall accuracy when large numbers are multiplied by each other. Addressing the research challenge of approximate multiplication in LUT-based FPGA systems, Ullah et al. proposed a 4x2 approximate multiplier as an elementary module for Xilinx 7-series FPGAs with optimized logic equations for each PP bit [163]. They further combined two 4x2 multipliers and formed a 4x4 approximate multiplier, and introduced additional approximation by discarding one of the two carry propagation signals. Their analysis also includes an evaluation of higher-order multipliers constructed using these basic multiplier blocks.

Approximate PP Accumulation: Various techniques have been proposed for the approximate summation of PPs in a multiplier. Mahdiani et al. proposed Bio-inspired Imprecise Computational blocks (BICs) and integrated this into the multiplier architecture for effective approximations [141]. These blocks consist of a Broken-Array Multiplier (BAM), which has a horizontal-broken level (HBL) and a vertical-broken level (VBL) that eliminates a desired number of bits horizontally and vertically from the PP tree. Therefore, this technique saves AND gates corresponding to the eliminated product bits and Full Adder (FA) cells that use these product bits, thereby saving area and power and reducing latency. Farshchi et al. extended the idea of BAM to the Booth multiplier [164] by keeping only the VBL and proposed broken-Booth multiplier (BBM) [165]. Error Tolerant Multiplier (ETM) proposed by Kyaw et al. splits the input operands into a multiplication part where higher order bits are multiplied precisely and a non-multiplication part with remaining lower order bits are approximated [166]. The non-multiplication part ignores carry propagation, and the input operand bits are inspected for "1" from MSB to LSB. From the position where one of the input bits is "1", the output bits of this non-multiplication part is set to "1". The Static Segment Multiplier (SSM) approach takes a number of consecutive bits from each input operand with a size greater than or equal to half of their size [167]. These segments can be chosen either from the MSB or LSB side and include leading ones bit. Finally, these selected segments are accurately multiplied. The experimental results showed that SSM could reduce overall energy consumption at the expense of a little loss in computational accuracy. The Accuracy Configurable Multiplier (ACMA) proposed by Bhardwaj et al. approximates the intermediate bits with a carry-in prediction for better power consumption and performance [168]. However, this approach renders the MSB part, which preserves the important information and the lower bits, which do not require extensive hardware resources, accurately. Similarly, the Dynamic Range Unbiased Multiplier (DRUM) identifies the leading ones in the n-bit operands, and a user-defined k number of bits are selected based on the desired accuracy [169]. In addition, the LSB of the selected parts are set to "1" for unbiased approximation and then use these k-bit operands for accurate multiplication. The Truncation and rounding-based scalable approximate multiplier (TOSAM) proposed by Vahdat et al. truncates each input operand based on their leading one-bit position to reduce the number of PPs [170]. Their implementation includes a small multiplier with inputs rounded to the nearest odd number to compensate for the error from the truncation.

Approximate counters and compressors are used in the accumulation of PPs in approximate multipliers. Lin and Lin approximated both carry and sum using a 4:2 counter in an inaccurate 4x4 Wallace multiplier [171]. Momeni et al. simplified 4:2 compressors for the Dadda multiplier with a relatively low error probability [172]. Similarly, Ansari et al. proposed an approximate 4:2 compressor for multipliers, and they improved it by encoding the inputs to form 4x4 multipliers [173]. Using these multipliers as basic building blocks, they formed larger 16x16 and 32x32 multipliers. Strollo et al. compared different 4:2 compressors used in low-power approximate multipliers with their novel extension, and larger multiplier circuits were designed using their proposal [174].

Functional Approximation: The functional approximation often simplifies the logical operation behind a multiplication function to improve hardware resource utilization. Redefines multiplication function, a Rounding-Based Approximate Multiplier (RoBA) uses the rounding of input operands nearest to the power of two [175]. A simple multiplication can be re-written as $A \times B = (A_r - A) \times (B_r - B) + A_r \times B + B_r \times A - A_r \times B_r$, whereas A_r and B_r are the rounded values. In RoBA, the first term $(A_r - A) \times (B_r - B)$ is omitted because the value of this term is small compared to the other terms. Due to the power of two rounding, the remaining multiplication operations can be computed with simple shift operations in hardware. Additionally, an efficient Kogge-Stone adder [176] and a simplified subtraction operation are also used for addition and subtraction terms. The experimental results show that the RoBA can effectively trade-off accuracy for the hardware resources. Extending the RoBA approach, Garg and Patel proposed Energy efficient Rounding-Based Approximate Multiplier (RBA) by simplifying the multipliers for further reducing implementation complexity at the expense of computation accuracy [177], and the efficacy of their approach was demonstrated using a Gaussian filter application.

Further approximating the multiplication function, Logarithmic Multipliers (LMs) are introduced, which replace the multiplication operation with a simple addition in the logarithmic domain. In early 1960, Mitchell proposed a logarithm-based algorithm for multiplication and division, which approximates logarithmic and antilogarithmic conversion [178]. A simple LM implementation requires leading one detector and encoders to use Mitchell's algorithm for each operand [19]. After a simplified logarithmic conversion, these operands are added, and the final multiplier product is obtained by converting the sum using an antilogarithmic converter. Many variants of LM have been proposed over the past decades, simplifying the operation or addition of error correction. These include a truncated logarithmic multiplier, which truncates the mantissa using a configurable width [179], an improved Mitchell-based logarithmic multiplier with an error correction term before the antilog conversion [180], a minimally biased multiplier (MBM), which combines the truncation and a constant error correction term [181], a reduced-error approximate logarithmic multiplier (REALM) with a correction coefficient table for each power of two intervals [182], and an improved logarithmic multiplier (ILM) by replacing the leading one detector with nearest-one detector corresponds to the closest power of two [183]. Additionally, Liu et al. proposed a set of approximate logarithmic multipliers (ALMs) by further replacing the adder with approximate adders [184]. Ebrahimi et al. proposed Leading-one Detection-based Softcore Approximate Multipliers (LeAp) for FPGAs, which efficiently utilizes 6-input LUTs and fast carry chains for an approximate logarithm calculator to implement Mitchell's algorithm [185].

A generic functional approximation has been performed on gate-level multiplier circuits using CGP to obtain different approximate multiplier variants in EvoApprox libraries [159, 160]. Together with approximated adders, EvoApprox8 proposed by Mrazek et al. includes 471 8-bit approximate multipliers [159]. Later, the extended EvoApproxLib also includes a total of 101 405 n-bit and mxn-bit approximate multiplier circuits [160].

2.1.3.4 Approximate Dividers

Approximate dividers are not frequently used as adders or multipliers in digital image processing applications. However, these components often consume considerable resources when they are used. A typical array divider uses a multiplexer and a subtractor cell to retain a partial reminder and generally requires n^2 subtractor cells for a $\frac{2n}{n}$ division. Several efficient implementations have been proposed to reduce the critical path at the expense of hardware overhead [19], and approximate computing further improves resource savings. Approximate dividers are generally classified based on subtractor cell approximations for array dividers, exact dividers with reduced bitwidth, and functional approximation of division operation.

In the first category, approximate unsigned nonrestoring divider (AXDnr) by Chen et al. used circuit simplification to approximate subtractors cells, and different AXDnr are proposed either by replacing accurate divider cells in a divider array with these approximated versions or by removing some divider cells itself [186]. Later, the same research group extended the divider cell, named approximate unsigned nonrestoring array divider cell (AXDCnr), which shows a better quality-resource trade-off [187].

Secondly, an approximate dynamic divider proposed by Hashemi et al. routes the selected bits of operands to a small accurate divider using a steering logic that dynamically detects the most significant bits of each operand [188]. They introduced approximations in the steering logic where each operand is approximated by truncation of some lower bits and showed a better average error-resource trade-off than the AXDnrs.

Similar to approximate multipliers, the third functional approximation uses Mitchell's algorithm that can simplify dividing operation by using a logarithmic and antilogarithmic conversion [178]. An approximate integer divider (INZeD) and approximate floating-point divider (FaNZeD) proposed by Saadat et al. are also employed Mitchell's algorithm with error-correction techniques [189]. A non-iterative high-speed division proposed by Low and Jong introduced a new antilogarithmic algorithm to merge multiple stages of Mitchell's algorithm into a single one for approximate division [190]. Analysis of their approach showed a better speed and area-delay product at the expense of some area overhead. An approximate hybrid divider (AXHD) proposed by Liu et al. combines a restoring array and logarithmic dividers to get a better quality-resource trade-off [191].

A division operation is simplified by a multiplication which can be obtained by rounding the divisor to a form of $\frac{2^{K+L}}{D}$, where K is the leading one position, and L and D are constants obtained by simulation with the lowest relative error in High speed yet Energy Efficient Rounding-based Approximate divider (SEERAD) approach [192]. Overall, the above transformation simplifies the division operation to some add and shift operations, and by varying the constants L and D, a better quality-resource trade-off can be obtained. Similarly, Imani et al. proposed Configurable

Approximate Divider (CADE) in which division operation is approximated with a subtraction of the input operand mantissa [193], and the accuracy can be configured at run-time.

2.1.4 Device or Transistor Level Approximations

The device or transistor level approximations are often performed globally on a device or modify the transistor behavior, which generally affects an entire device. Voltage overscaling is a commonly used approximation technique in this category, and such approximations are often non-deterministic in behavior.

2.1.4.1 Voltage Overscaling

Similar to the memory level approximations, the voltage overscaling technique reduces the supply voltage more than a level typically safe for a given clock frequency. Therefore, the overscaled voltages cause the critical paths to violate the clock period and introduce errors. Many approaches have exploited this for an energy-efficient system design. Hegde et al. used voltage overscaling in which an introduced error is compensated by a low complexity error control that exploiting the signal statistics [194]. Similarly, Marković et al. proposed a sensitivity-based optimization framework that uses supply and transistor threshold voltage as tuning knobs for determining the energy-delay trade-off [195]. Another approach proposed by Kahng et al. enlarges the range at which the error is acceptable using a technique called power-aware slack redistribution [196]. This technique iteratively finds frequently used non-critical paths and shifts to maximize power benefit for a predefined error rate. Mohapatra identified coarse-grained meta-functions that are iteratively executed across multiple cycles in an application and applied voltage overscaling for a better energy-quality trade-off [148]. Additionally, a dynamic segmentation of components in the meta-functions reduces the critical path, thereby allowing aggressive voltage overscaling. In an attempt to develop a framework for fast and accurate simulation of voltage overscaling, Zervakis et al. introduced VOSsim for the estimation of error-power approximate design trade-offs [197]. Later, another research by Zervakis et al. extended the idea of voltage overscaling to different voltage islands and minimize power consumption by grouping and assigning different supply voltage for approximate circuits in the islands [31]. Since the errors introduced by the voltage overscaling methods are non-deterministic, special care must be given during the designing phase to ensure the quality-threshold requirement in an approximate computing application.

2.2 General-Purpose Approximate Computing Methods

In past years, multiple research has already been published addressing the problem of arbitrarily approximating circuits or multiple system components or combining specific approximations in an application. Many of such general-purpose design flows consider different approximation techniques in the same or multi-level abstraction, requiring only a little or no designer intervention. Similar to our proposed AxCGA, some of these approaches are proposed directly for FPGA designs, while most of them are presented for ASIC designs. However, many of these ASIC-based methods, such as high-level synthesis (HLS)-based approaches, can either be partially or fully translated for

Table 2.1: Overview of state-of-the-art general-purpose approximate computing design flows

Technique	Approximation Methods	Multi-Level Abstraction	Error Control	Search Methods	Search Type	Considered Objectives*	Target Platform
SALSA [198] cp. Section 2.2.1	Functional approximation	No	Bounds	Simple iteration	Single-objective	A, EP AEQ	ASIC
SASIMI [20] cp. Section 2.2.2	Functional approximation	No	Bounds	Hill- Climbing	Single-objective	A, DLP EP, AEQ	ASIC
ABACUS [21] cp. Section 2.2.3	Loop perforation Approximate arithmetic Functional approximation Precision scaling	Yes	Bounds	Greedy algorithm	Single-objective	A, EP AEQ	ASIC FPGA
ASLAN [199] cp. Section 2.2.4	Functional approximation Precision scaling	No	Bounds	Hill climbing	Single-objective	EP, AEQ	ASIC
CGP [201] cp. Section 2.2.5	Approximate arithmetic	No	Bounds/ None	Genetic algorithm	Single-/Multi-objective	A, DLP EP, AEQ	ASIC FPGA
GRATER [207] cp. Section 2.2.6	Precision scaling	No	Bounds	Genetic algorithm	Single-objective	A AEQ	FPGA
Xu and Schafer [30] cp. Section 2.2.7	Task skip, Precision scaling Approximate arithmetic	Yes	Bounds	Greedy algorithm (partially)	Single-objective	A, DLP EP, AEQ	ASIC FPGA

* A: Area, EP: Energy/Power, DLP: Delay/Latency/Performance, AEQ: Accuracy/Error/Quality

Table 2.1: Overview of state-of-the-art general-purpose approximate computing design flows (Continued)

Technique	**Approximation Methods**	**Multi-Level Abstraction**	**Error Control**	**Search Methods**	**Search Type**	**Considered Objectives***	**Target Platform**
SCALS [208] cp. Section 2.2.8	Functional approximation	No	Bounds	Simple iteration	Single-objective	A, DLP AEQ	ASIC FPGA
Zervakis et al. [31] cp. Section 2.2.9	Voltage overscaling Approximate arithmetic Precision scaling	Yes	Bounds	Binary search Greedy algorithm	Single-objective	EP AEQ	ASIC
autoAx [209] cp. Section 2.2.10	Approximate arithmetic	No	None	Hill climbing (modified)	Multi-objective	A, DLP EP, AEQ	ASIC
CIRCA [210] cp. Section 2.2.11	Functional approximation Precision scaling	No	Bounds	Hill climbing	Single-objective	A, DLP EP, AEQ	ASIC FPGA
ApproxFPGAs [211] cp. Section 2.2.12	Approximate arithmetic	No	None	Hill climbing (modified)	Multi-objective	A, DLP EP, AEQ	FPGA
AxHLS [212] cp. Section 2.2.13	Approximate arithmetic	No	Bounds	Tabu search	Single-objective	A, DLP EP, AEQ	ASIC FPGA
E-IDEA [213] cp. Section 2.2.14	Loop perforation Precision scaling Approximate arithmetic	No	None	Genetic algorithm	Single-/Multi-objective	A, EP AEQ	ASIC FPGA

* A: Area, EP: Energy/Power, DLP: Delay/Latency/Performance, AEQ: Accuracy/Error/Quality

FPGAs designs. The following sections describe different state-of-the-art approximate computing design flows that primarily aim to integrate multiple approximations in an application. The general characteristics of these methods are summarized in Table 2.1. Each design flow is further analyzed based on seven different criteria: type of approximation methods that can be integrated, abstraction levels where the approximation methods can be applied, type of error control mechanisms that consider quality-threshold, search method that identifies the optimal parameter configuration, claimed or analyzed design objectives, and finally, the design platforms where these design flows can be targeted.

2.2.1 SALSA: Systematic Logic Synthesis of Approximate Circuits

Systematic methodology for Automatic Logic Synthesis of Approximate circuits (SALSA) is an approximate circuit synthesis framework that takes register-transfer level (RTL) description of a circuit and error bounds as user inputs and saves hardware resources such as area or power by adhering to these error bounds [198]. The core methodology of SALSA lies in the formulation of a Quality Constraint Circuit (QCC), composed of the original and an approximate circuit and a quality function. The quality function output is computed from these two versions of circuits and returns "1" if the approximate circuit fulfills the error bounds, otherwise returns a logic "0". A reliable approximate synthesis using SALSA is considered when this output should be "1" for all input combinations. The approximation circuit is iteratively evolved during the synthesis process using Observability Don't Care (ODC) concept while keeping the quality function output at "1". The ODCs are the input values for which the primary outputs of a circuit are insensitive to the output of a considered circuit node. Therefore, the quality function output of QCC will also remain unaffected. Such ODCs are called Approximation Don't Cares (ADCs) in SALSA and give a set of inputs that an output bit of an approximate circuit and thereby the QCC output are insensitive. By using these ADCs as external don't care, a standard don't care based synthesis technique synthesis an approximate circuit. In this approach, a DSE is missing for an effective trade-off analysis between the objectives. Instead, for each user-defined error bounds, SALSA identifies ADCs and synthesizes using standard tools to estimate the hardware resources such as area or power.

2.2.2 SASIMI: A Unified Design Paradigm for Approximate Circuits

Substitute-And-SIMplIfy (SASIMI), proposed by Venkataramani et al. judiciously identifies signal pairs in a circuit with a high probability of having the same value and substitutes one for the other [20]. Such a target signal is successively substituted by a substitute signal, which might be logic zero, logic one, other signals, or their compliments until the target error constraints are met. SASIMI takes the original circuit and target error as inputs from a user, and the best target signal-substitute signal pair is identified in each iteration. The error is computed in each iteration as a weighted sum of normalized logic deletion potential and downsizing potential. These potential values are estimated from the size of logic removed from substitution and the signal arrival times with and without substitutions, respectively. The iteration moves towards the next level only if the error constraints are met, and this DSE resembles a hill climbing algorithm. SASIMI can also synthesize quality configurable circuits that can dynamically operate at different accuracy levels depending on the application requirements. Even though an area overhead is introduced by the runtime substitution, clock expansion, and quality selection circuits, SASIMI could achieve considerable area savings compared with the original circuits. Since the focus of SASIMI is on the functional approximation by logic simplification, this approach can be translated well to the FPGA designs. However, it cannot maximize the potential savings due to the architectural differences.

2.2.3 ABACUS: Approximate HLS using Approximate Functional Units

Automated Behavioral Synthesis of Approximate Computing Systems (ABACUS) proposed by Nepal et al., is an approximate circuit synthesis method from the behavioral description of a circuit [21]. ABACUS first generates an abstract synthesis tree (AST) from the behavioral description. Thereafter, approximate design variants are obtained by applying transformations to this AST. Five different transformations are employed for the approximation: 1) data type simplification, which involves truncation of bits or resetting a number of LSB to zero, 2) operation transformation, in which arithmetic operations are replaced with corresponding approximate versions, 3) arithmetic expression transformations, where near similar structures share a transformed similar structure, 4) variable to constant substitution, and finally 5) loop transformation by perforation and iteration skipping. An iterative stochastic greedy algorithm is performed to explore resulted design space, and the fitness value of each parameter configuration is computed as a weighted sum of power, area, and accuracy. Additionally, ABACUS keeps individual resource usage and accuracy from these iterations and finally forms Pareto-optimal solutions based on these values. However, ABACUS requires both simulation and synthesis to determine the fitness of a parameter configuration during the greedy search. Therefore, using ABACUS is time-consuming for large applications with many parametrizable approximations.

2.2.4 ASLAN: Synthesis of Approximate Sequential Circuits

Automatic Methodology for Sequential Logic ApproximatioN (ASLAN), proposed by Ranjan et al., is an automated approach for the synthesis of approximate sequential circuits [199]. Similar to SALSA, this approach also relies on a sequential QCC circuit, which includes an original sequential circuit, an approximate sequential circuit, and a Quality Evaluation Circuit (QEC). The QEC evaluates the quality and generates two output bits, such as quality (Q) and quality valid (V). To ensure the quality constraints, ASLAN sequentially check whether the V bit is true for all the possible QCC states, and if it is true for all the states, then the Q is also set as true. In ASLAN, combinatorial components which are amendable for approximation are identified, and each of this component is approximated with existing techniques to determine a local energy-quality trade-off. The ASLAN algorithm heuristically identifies suitable candidates to approximate using these pre-generated trade-offs. Thereafter, the identified candidates are sorted, and the first component on the sorted list is replaced by its approximate version. Afterward, this approximate circuit is verified for quality constraints using QEC and moves to the next iteration only if the quality constraints are met. Otherwise, the current approximation is ignored, and the algorithm selects the next one from the sorted list for the following approximation. This approach uses functional simplification with SALSA and precision scaling as the approximation techniques. Even though this approach identifies the overall energy-quality trade-off from the identified approximated versions, the search approach resembles a local hill climbing algorithm. Therefore, an extensive and global quality-resource trade-off exploitation is not feasible with this approach.

2.2.5 CGP: Approximate Computing using Cartesian Genetic Programming

Cartesian Genetic Programming (CGP) is an evolutionary algorithm suitable for various applications, including digital circuit design and optimization [200]. Different researches have been published which use CGP for functional approximation of digital circuits [201]. Therefore, this section provides a high-level overview of CGP principle.

In CGP, a circuit is modeled as a two-dimensional array of nodes with a number of rows and columns, and these nodes can be elementary logic functions, transistors, or arithmetic components. Each node in the circuit can be connected either to a circuit input or to an output of certain previous nodes. Such an array is encoded using three integers for every node, where the first integer represents the functionality, and the last two integers represent the inputs to the node. The encoded representation is called a genotype or chromosome in CGP, and each integer represents a typical gene. Additionally, a number of integers in the last part of the chromosome represents the output connections to the circuit. In general, the size of the genotype is always constant, while the phenotype, which represents the actual circuit implementation, can have different sizes during the CGP optimization. The CGP employs a $(1+\lambda)$ evolution strategy and a mutation operation that randomly replaces a number of encoded genes with other valid ones. The mutated chromosome is compared with the reference circuit without approximation for the number of gates representing the resource utilization and the error computation with every possible input stimulus. The evolutionary operation with the mutation iteratively continues until a termination condition is satisfied.

Approximations can be integrated into CGP design flow mainly using three different techniques. The first resource-oriented method searches for a circuit with a target of k_i gates, where $k_i < K$ and K is the number of gates in a reference circuit [202]. Therefore, this approach is more resource controllable for a designer. Thereafter, the CGP is executed several times with different k_i for the trade-off estimation. In the second error-based method, different error values are estimated using the output of fully functional and approximate circuits during the evolutionary iteration, and then find a chromosome with minimal gate count which satisfies a certain user-given error rate [203]. Finally, the third type of method replaces the single-objective optimization with multi-objective NSGA-II to identify a good compromise between error, area, and delay [204]. Even though most of the CGP approaches use gate-level design for approximation and optimization, some approaches extend this to the FPGA system by synthesizing the obtained design from the initial gate-level approximation and optimizations [205, 206]. In general, these methods are directly proposed for the circuit level approximation and are not suited for methods that target other abstraction levels, such as algorithm or memory levels.

2.2.6 GRATER: An Approximation Workflow for FPGA Acceleration

Lotfi et al. proposed an approximation workflow, GRATER, specifically targeting accelerators in FPGA applications [207]. The core methodology involves a source-to-source compiler that applies precision scaling to the input kernels using a novel optimization technique. The GRATER compiler uses OpenCL kernel transformation for integrating the approximations and takes an exact OpenCL kernel, a set of input test cases, and quality metrics as user inputs. Thereafter, GRATER automatically identifies variables in a way that generates separate kernels for checking the compliance of each variable. In each kernel, the precision of one variable is lowered by a level, and the precision of all other variables is kept unchanged to determine the quality loss with the help of profiling feedback. For example, GRATER uses four precision levels for a variable with data type change from float to char, i.e. {4, 3, 2, 1} for {float, int, short, and char} respectively. If the quality loss is acceptable, that particular variable is added to a safe-to-approximate variable list. For all the variables in this list, the lowest precision is computed in a similar way by generating a separate kernel and varying the precision from the lowest until the quality result falls in the acceptable region. Using these bound of each variable, a single objective GA-based DSE is performed for minimizing the FPGA resource utilization, which is estimated from the precision level of variables. The DSE stops when the best chromosome with the lowest fitness, i.e., resource on FPGA, is determined. In this way, GRATER enables more kernels to run parallelly on target FPGA platforms. However, GRATER uses only precision scaling as the approximation technique. Therefore, this approach is missing the possibility of exploiting maximum benefit from multiple approximations on different abstraction levels.

2.2.7 Approximate Computing Optimizations from Behavioral to Gate-Level

Xu and Schafer proposed a multi-level approximate computing design flow for HLS [30]. Their approach involves a four phase approximation, and each phase is dedicated to approximations on a specific abstraction level. Initially, a designer has to specify behavioral description of a design, provide a library of exact and approximate functional units for different bitwidths, an application-specific error bound, and finally, input stimuli with chosen data distribution. In the first phase, all the loops are identified and automatically unrolled at the software level based on a specified pragma in C. However, depending on the unroll factor specified by the pragma, this unrolling can be done either partially or fully or even not unroll at all. Subsequently, the unrolled code is profiled using the input stimuli, and infrequent code lines are pruned away. In addition, a signal-to-signal and a signal-to-constant simplifications are applied, and profiling of internal signals helps to prune parts of circuits that need to calculate some internal signals. Thereafter, each functional unit is replaced with an individual approximate functional unit from the library, and the modified behavioral description is characterized using HLS in terms of resources such as area, delay or latency, and error estimated from cycle-accurate system C models. A greedy algorithm iteratively adds multiple replacements, and a Pareto-optimal trade-off is estimated for minimizing weighted sum of error and resource consumption. Furthermore, dependency analysis is also performed to check the relation between each functional operation, and the dependent operations require HLS of all functional unit combinations that have an error below the target error. In the third phase, a variable-to-variable and a variable-to-constant substitutions at the RTL are performed, together with forcing of different signal bits to logic "1" or "0". Such substitution in the RTL is beneficial since all the internal signals are exposed for further approximations, in fact, a synthesis and gate-level simulation for profiling are time-consuming. A final stability analysis is also performed on the selected approximate circuits at the gate level using input stimuli with various distributions. In general, this approach involves the sequential application of the different approximation methods in multiple abstraction levels and does not include any resource modeling for fast profiling. Additionally, a clever DSE method is also missing, especially in the case of parameter dependency. Rather, this approach requires HLS of all approximate functional unit combinations that meet the target error bound.

2.2.8 SCALS: Statistically Certified Approximate Logic Synthesis

Liu and Zhang proposed Statistically Certified Approximate Logic Synthesis (SCALS) technique using stochastic optimization approach. The SCALS synthesizes approximate designs with user-specified error constraints and input distributions after mapping the technology-independent logic gates for a specific technology library such as LUT-based FPGAs or ASIC-based standard cells [208]. Their approach starts with normal technology mapping, and then the mapped netlist for a target technology is divided into several sub-netlists for logic optimization. An iterative logic optimization is performed on each sub-netlist in isolation, together with a set of exact and approximate transformation moves. The exact transformations include logic operations such as balance, rewrite, and refactoring that does not change the functionality, whereas they optimize the gate count or balance the logic depth. However, the approximate transformations simplify the logic and might generate errors in the output. The approximate transformation moves include reduce, flip, or add, which randomly removes or inverts an input of a logic gate or adds a random logic with arbitrarily selected inputs. Initially, a transformation move is selected with a certain probability, and after each transformation, a new netlist is mapped to the target technology to estimate the area. Similarly, the netlist is simulated using inputs with user-given error distribution to compute the error metrics. A single quality value is estimated based on this area and error. Thereafter, the quality metric is compared to the previous iteration using the Markov Chain Monte Carlo method to determine whether the current move is acceptable. Although this approach can be directly employed to FPGAs, the single objective optimization limits an effective trade-off analysis.

2.2.9 Accelerator Synthesis Under Voltage Island Constraints

A multi-level approximate accelerator synthesis under voltage island constraints proposed by Zervakis et al. considered several approximate arithmetic components together with a voltage over-scaling technique [31]. Their approach initially creates a multi-level approximate arithmetic library which includes approximations at the algorithmic level, such as perforation or truncation, and at the circuit level, such as functional approximation. Each of the library components also contains a Pareto-optimal implementation with error-power characteristics. For the approximate accelerator synthesis, a designer has to specify a behavioral description and a DFG of the accelerator, global error bounds, and a number of voltage islands which has different degree of voltage overscaling. In this approach, the global error is distributed to different DFG nodes at first. Thereafter, a selection of appropriate approximation techniques and their configuration is applied to each DFG node. This second phase includes generating a number of random DFG configurations by replacing accurate arithmetic components in each node with approximate ones from the library. Each of these DFG instances is synthesized and simulated for the power and error values. Using these values, an ANN is trained to model the error values. The overall power of a DFG instance is estimated as the sum of power dissipation over each node. With these error and power models, they estimated the fitness of all DFG configurations and finally determined optimal DFG using a binary search technique. As a final step, the voltage islands are formed iteratively to limit the number of different supply voltage levels using a greedy algorithm. This approach originally targets ASIC designs, which support different supply voltages to individual components. However, the voltage overscaling with locally different voltages is not possible on current commercial FPGAs, and therefore, this approach can only be partially applied to the FPGAs.

2.2.10 autoAx: Automatic Design Space Exploration and Circuit Building

An automatic DSE and circuit building methodology using libraries of approximate components (autoAx) proposed by Mrazek et al. aims to combine multiple approximate circuits from state-of-the-art approximate component libraries into a target design [209]. Their approach proposed a three-phase methodology for searching, selecting, and combining multiple approximate methods in an application. For a target application, the approximate circuits from the state-of-the-art libraries are preprocessed initially to remove all the irrelevant circuits. This preprocessing phase profiles application-specific error and hardware cost for each approximate component using benchmark data, and only a subset of components that lies in the Pareto front is preserved for further phases. In the second phase, the computational models for the quality of results and hardware cost are trained using machine learning algorithms after synthesis and simulations of hundreds or thousands of randomly selected configurations of a target application. A fast and exhaustive model-based DSE is performed in the next step using these models to identify Pareto-optimal solutions. Due to numerous possible parameter combinations, they proposed an iterative heuristic algorithm, similar to a hill climbing approach, with a restarting mechanism to avoid stagnation in local optima. The DSE Pareto solutions are then simulated and synthesized for the final Pareto front, enabling a designer to select reliable approximate designs in an application. The autoAx approach is specifically targeting ASIC platforms and considers only circuit level approximate arithmetic components, which prevents exploiting benefits from multiple abstraction layers. Additionally, the modeling phase requires a time-consuming synthesis of multiple configurations for each target design. However, their DSE approach is scalable with a large number of approximation parameters, capable of estimating the quality-resource trade-off, and can consider multiple trade-off objectives. Therefore, this thesis uses the autoAx DSE approach to compare the performance of the DSE approaches proposed for AxCGA.

2.2.11 CIRCA: Modular and Extensible Framework for Approximate Circuits

A concept for a modular and extensible framework for approximate circuit generation, named CIRCA, was proposed by Witschen et al. uses a three-phase operation to approximate circuits from a library of approximate components [210]. The first input stage processes user-provided information such as Verilog description of a circuit and a configuration file. The configuration file includes information for the next phase, such as target metrics, quality constraints, and employed functionality. Additionally, a user can specify an input vector for quality assurance as well. The second stage, called Quality Assurance, Approximation, Estimation, and Search Space Exploration (QUAES), is the central part of the framework. A candidate, which is an annotated subcircuits in the Verilog code, can be replaced with different approximate variants from the library. The search space exploration is iteratively explored by expanding circuit configurations using approximate candidates and evaluating them for quality error and hardware resources such as area, delay, and power. The search algorithm determines the configuration which has to be selected and expended for further steps. Subsequently, this configuration is validated to check whether the quality constraints are met. If the constraints do not meet, the search algorithm picks the next best configuration for the validation, and this process will continue until no more valid configurations exist. Even though this approach claims that it supports a wide range of search algorithms, the experiments use hill climbing for the search space exploration. All the valid configurations identified in the QUAES stage are stored, and the final output stage postprocess these configurations for a Pareto front or the best target circuit with given quality constraints. Similar to other single-objective search approaches, an effective quality-resource trade-off analysis is therefore missing in this approach.

2.2.12 ApproxFPGAs: ASIC-Based Approximate Arithmetic Components for FPGAs

ASIC-based approximate arithmetic components for FPGA-based systems (ApproxFPGAs) proposes an automated arithmetic approximation design flow for FPGA designs [211]. The approximation techniques proposed for ASICs can often be employed on the FPGAs to save resources. However, due to the architectural differences, it causes asymmetrical gain on FPGAs for Pareto configurations identified for ASIC. The ApproxFPGAs is an extension of autoAx proposed by Mrazek et al. [209] to effectively approximate FPGAs using the state-of-the-art approximate component libraries proposed for ASICs. Similar to autoAx, the preprocessing phase removes all irrelevant components for a target application from the approximate component library. However, the accurate resource estimation on FPGA is time-consuming due to the extensive synthesis time of entire library components for FPGA. Therefore, the ApproxFPGAs uses statistical and machine learning-based models for determining hardware cost and identifying preprocessed Pareto-optimal approximate components. In the first part, hardware resource models are trained with synthesis results of a randomly extracted subset of approximate components from the state-of-the-art libraries. The resource models for FPGAs are trained for power, latency, and area in terms of LUTs. Due to limited fidelity of these models, initially, a pseudo Pareto front is identified using the models for varying bitwidth of approximate arithmetic components. Thereafter, the components that lie in the pseudo front are synthesized for the target FPGA to generate the accurate Pareto-optimal approximate components. Finally, to generate an approximate target application, autoAx methodology is applied by replacing the preprocessed Pareto-optimal ASIC library with the new FPGA library and estimator for DSE models with FPGA resource estimators. Even though the ApproxFPGAs construct an FPGA-based library for approximate arithmetic components and directly targets FPGA applications, the subsequent autoAx DSE requires synthesis and simulation of multiple parameter configurations of target applications to train the application-specific models.

2.2.13 AxHLS: DSE and HLS of Approximate Accelerators

Castro-Godínez et al. proposed AxHLS, an automated framework for HLS of approximate accelerators using approximate arithmetic functional units such as adders and multipliers [212]. Their framework uses a library of approximate functional units and a set of analytical models named AxME, which is developed by characterizing individual components in the library for hardware resources such as area, dynamic and static power, and delay. Based on analytical studies, they proposed simple resource models for target applications using individual approximation components involved in the application. The total area of a target system is approximated as a sum of the area from individual system components. Similarly, the total static power is approximated as a sum of the static power of individual components. The dynamic power is estimated as a sum of individual dynamic power normalized with the dynamic power of the accurate design without any approximations. The delays are modeled using operations located in the critical path of the circuit. A tabu search based optimization approach is used in the proposed DSE methodology, named DSEWam. The DSEWam iteratively replaces the approximate components based on a combined fitness value from resource models and forms a Pareto front with corresponding error values. The proposed methodology is integrated into an HLS design flow to generate approximate accelerators from a behavioral description in C and demonstrated using a few image filter applications. As the core of all these approaches lies in the combination of different arithmetic units, this method is also restricted to the circuit-level approximation. In addition, the DSEWam is a single-objective local search metaheuristic, limiting the possible exploitation of quality-resource trade-off.

2.2.14 E-IDEA: Multi-Objective Application-Driven Approximate Design Method

Barone et al. proposed a two-level multi-objective application-driven approximation method, E-IDEA, that combines multiple approximations in different abstraction levels [213]. The framework takes a circuit in C/C++, a set of approximate operators called mutator, and fitness functions which comprise error and gain metrics as inputs and return a set of non-dominated configurations obtained with best mutator configurations. The first phase of the framework is a source-to-source manipulation tool, named Clang-Chimera, that analysis the circuit to apply approximations using a Match and Mutate operation. The Match identifies the code that can be replaced with approximate operators using the Mutate operation. This approach considers multiple approximations such as loop perforation, precision scaling for both floating-point and integer arithmetic, and approximate arithmetic components. The second phase, Bellerophon, identifies the best approximation version of a given C/C++ code according to target design goals. Bellerophon explores the design space using NSGA-II with different possible mutator configurations and forms a set of Pareto-optimal solutions trading off the defined objective functions. This approach is demonstrated using different target designs and identified Pareto-optimal solutions for these designs. Additionally, an HLS is also used to get the actual hardware implementations of the circuit. Therefore, this approach can be employed for both software and hardware implementations, including FPGA designs. However, this approach cannot consider any architectural features of FPGAs in optimizations. Additionally, even though this framework performs typical approximations on different abstraction levels, combining multiple approximations on the same abstraction level or multi-level is missing.

2.3 Summary and Discussion

This chapter reviews various state-of-the-art approximate computing techniques presented in the literature. These techniques are broadly classified as single-purpose or general-purpose based on the scope of approximation in an application, and the core concept of each technique is explained with suitable examples. Initially, the single-purpose techniques that approximate a specific type of system component or functionality are described and further classified into four categories based on the targeted level of abstraction in an application, such as algorithm or software level, memory level, circuit level, and device or transistor level. Thereafter, the general-purpose design flows that arbitrarily approximate a complete circuit or combine multiple approximation techniques in a single system are discussed. These design flows are analyzed further based on the considered approximation strategies and their target abstraction levels, employed search heuristics and search type, the number of objectives as single or multi-objective, claimed or considered trade-off objectives, and finally, the target computing platforms.

This thesis aims to develop a framework for approximate image processing applications on FPGAs that can effectively trade-off application quality for hardware resources. The majority of the analyzed single-purpose methods are proposed for the ASIC designs. However, most of such methods translate well for the FPGAs, even though the approximation benefits are not linearly scaled. Algorithmic techniques typically benefit irrespective of the target platform, including FPGAs, although many researchers did not estimate the benefits directly on FPGAs. For example, table-based approximation methods storing only a subset of data can reduce memory usage in DRAM in ASIC-based designs and can potentially save DRAM usage or the BRAMs in FPGA-based designs which have a different architecture. Similarly, a circuit level approximation which reduces gate count or length of the critical path can often save some LUTs and registers or reduces latency in FPGAs as well. Since external memories are often coupled with FPGAs, memory level approximations can also be integrated into an approximate FPGA design. Additionally, global voltage overscaling techniques can also be applied to FPGAs in a controlled way. Therefore, the approximate component library as a part of our proposed AxCGA framework adapts multiple of these techniques, which are parametrizable for quality-resource trade-off and deterministic to reproduce similar error behavior in an application.

Many real-world signal and image processing systems with multiple processing stages can integrate different approximations simultaneously on various abstraction levels. Therefore, single-purpose methods, such as algorithmic approximations or circuit-level approximations alone, can not exploit all of the potentials of approximation. While most of these single-purpose techniques in literature target the characterization and implementation of a specific type of system component or functionality, the general-purpose approximate computing design flows are introduced that arbitrarily approximate circuits or combine multiple approximation techniques on an application level. The general-purpose methods proposed in the literature are demonstrated their potential to integrate approximations into error-resilient applications with specific examples. Some of these approaches are proposed for the FPGAs applications, and many others proposed for platforms like ASIC, CPU, or GPU might translate well for FPGA designs. However, these approaches do not often include multiple significant attributes to exploit the maximum quality-resource trade-off, instead relying on some essential features.

In real-world applications, the choice of approximation technique depends strongly on the characteristics of the target application. Depending on the specific application, different methods might be most beneficial for a better quality-resource trade-off. For example, when signal calibrations or complex non-linear functions are implemented using pre-calculated look-up tables, table-based approximations are highly efficient, especially for reducing memory consumption. On the other hand, systems that perform many linear calculations benefit most from approximate arithmetic units or precision scaling. Therefore, a careful selection of specialized single-purpose techniques tailored to an application is necessary for a better quality-resource trade-off in real-world applications. In addition, the design flows should offer the possibility to support approximation techniques on multiple abstraction levels in an application as well. In literature, the design flows such as ABACUS [21], Xu and Schafer approach [30], and Zervakis et al. approach [31] offer the possibilities of combining multiple approximations on different abstraction levels, among them the first two approaches can be adopted for FPGAs designs. However, combining such multiple approximations often exponentially increases the possible configurations with the number of exposed tuning parameters from each individual method. Therefore, an efficient design space search technique is required in this case to identify the possible quality-resource trade-off. Many state-of-the-art design flows, including ABACUS, Xu and Schafer, and Zervakis approach, rely on a single objective search technique and can not perform an extensive trade-off analysis between conflicting design objectives. During the search, such single-objective optimization approaches often combine multiple design objectives into a single value with specific weights and consider only the convergence of this combined values to analyze the search performance. Therefore, these approaches cannot take into account the diversity of identified designs to form Pareto solutions for extensive trade-off analysis. However, the state-of-the-art design flows such as CGP [201], autoAx [209], ApproxFPGAs [211] and E-IDEA [213] consider multi-objective optimization for an effective trade-off analysis.

Addressing the shortcoming of the existing approximate computing works, this thesis presents an AxCGA framework that considers multiple relevant features to perform extensive trade-off analysis in approximate computing applications. The proposed AxCGA framework can simultaneously combine multiple approximations in various abstraction levels in an application and explore complex design spaces using a multi-objective genetic algorithm. We adopted the search technique in autoAx to explore the approximation parameters from multi-level abstractions and compared the performance of the autoAx DSE with AxCGA approach. Additionally, our novel search technique, ROI-NSGA-II, enhances both the diversity and convergence of explored Pareto solutions and helps a designer to make suitable design decisions.

CHAPTER 3

AxCGA: A DSE Framework for Approximate Computing Using Genetic Algorithm

This chapter describes our proposed framework AxCGA, a DSE framework for approximate computing on FPGA using GA, which effectively explores the design space exposed by combined approximations on different abstraction levels in FPGA-based applications. The core methodology has previously been published in "Model-Based Design Space Exploration for Approximate Image Processing on FPGA" @IEEE 2020 [23] and extended in "Model-Based Design Space Exploration for FPGA-based Image Processing Applications Employing Parameterizable Approximations" @Elsevier 2021 [29] with an additional case study. The optimization approach employed in AxCGA is further described in "Parameter Optimization of Approximate Image Processing Algorithms in FPGAs" @IEEE 2020 [27]. The figures, texts, and structure from these publications are adapted and extended for this thesis.

3.1 Introduction

Digital image or signal processing applications successively process the incoming data acquired either with sensor systems or from a preprocessing stage with multiple operations to achieve the desired application outcome. For example, an image filter or a color space conversion application involves a number of arithmetic operations such as multiplications and additions. Therefore, such an application offers the possibility to replace these accurate operations with their approximate counterparts, which expose parameters to tune their degree of inaccuracy. Additionally, the signals that connect these operations can also be subjected to the approximations, such as precision scaling. Combining such parametrizable multiple approximations in an application can deliver a better quality-resource trade-off for designers to make suitable decisions.

A careful approximation of various operations at multiple abstraction levels is required to maximize the benefits of approximate computing on an application level. Depending on the characteristics of these operations, typical single-purpose approximate computing methods have to be applied to trade in the accuracy for resource benefits effectively. However, this leads to a multi-objective DSE problem for which brute-force searches become infeasible as the design space size grows exponentially with the number of employed methods. Therefore, an efficient and effective search method is required to explore such a design space with fewer evaluations of possible solutions for resource usage and quality degradation. Additionally, the DSE approach has to avoid the time-consuming synthesis of the system for each parametrization and consider the architectural characteristics of the target platform for fast and reliable resource estimation. Furthermore, for a designer to make valid choices regarding acceptable quality degradation, the employed quality model should be suitable for the targeted application and interpretable for the designer. In general, the requirements for a successful approximation of an FPGA-based application can be summarized as follows:

- combining suitable approximations at multiple abstraction levels
- an efficient and effective multi-objective search to estimate the resource-quality trade-off
- application-specific quality models and fast and simple FPGA resource models are required

Many approximation methods focusing on different embedded computing hardware have been proposed, comprising single-purpose methods and general-purpose design flows that approximate multiple operations as described in Chapter 2. Many of these general-purpose design flows, either directly proposed for FPGAs or translating ASIC based methods to FPGAs, can be employed at an application level to approximate FPGA designs. However, none of these design-flows address all the primary requirements to exploit the maximum approximation benefits.

Approximate computing design flows proposed by Xu and Schafter [30], Zervakis et al. [31] and ABACUS [21] consider multiple approximations on different abstraction levels in an application. However, none of these approaches employs a multi-objective optimization approach for an efficient and effective quality-resource trade-off estimation. Additionally, Zervakis et al. employed voltage overscaling with locally different voltages, which is not applicable on current commercial FPGAs. Similarly, ABACUS requires a time-consuming synthesis of system configurations for resource estimation during each search iteration. The approach proposed by Xu and Schafter works with HLS and can not apply to established FPGA-based design flows that exploit the architectural characteristics of a target platform.

State-of-the-art approaches often primarily consider the cogency of approximate computing approaches. However, scrutinizing the effectiveness of DSE problem from an optimization perspective is yet missing. A single-objective DSE requires defining suitable weights for conflicting objectives to form an aggregated cost function and often makes the trade-off determination a challenging task. Instead, a multi-objective DSE bypass such a task and can effectively determine the trade-off. Considering various multi-objective state-of-the-art DSE approaches proposed for approximate computing, approximate computing using CGP [201] and E-IDEA by Barone et al. [213] uses NSGA-II for optimizing approximate parameter configuration and identifying the trade-off. Similarly, autoAx by Mrazek et al. [209] and ApproxFPGAs by Prabakaran et al. [211] employ a modified hill climbing algorithm that considers multiple objectives during the search to form Pareto configurations. In fact, these methods consider only circuit-level approximations, which prevent the exploitation of approximation benefits on other abstraction levels. The CGP approximates an entire gate-level circuit, and E-IDEA approximation is coupled with the HLS design flows. Therefore, these approaches do not take into the architectural characteristics of

FPGAs during the approximations. Similar to the DSE methodology employed in E-IDEA, our proposed AxCGA is also using NSGA-II for the exploration of the design space. However, NSGA-II based DSE in AxCGA was introduced already before E-IDEA was originally published. Even though autoAx and ApproxFPGAs consider only approximating arithmetic components, the search spaces explored in their demonstrated applications are complex due to a large number of approximate variants they consider. In addition, the DSE in their approaches is able to identify a wide range of Pareto-optimal solutions. Therefore, this work uses the DSE approach employed in autoAx/ApproxFPGAs to compare our proposed AxCGA and further denoted as autoAx DSE in the following sections.

Even though many of the ASIC-based approximations can trade-off resources for application quality on FPGAs, the approximation benefits do not linearly translate to the FPGAs. An optimal approximation parameter conflagration on ASIC does not necessarily be optimal in FPGAs. Therefore, approximate computing design flows that consider architectural characteristics of FPGAs can better exploit the resource-quality trade-off in an application. Additionally, a common characteristic of many state-of-the-art approximate computing designs flows is that they use basic signal difference metrics such as an absolute or relative error, an error rate, or a hamming distance for the quality estimation. While these metrics are capable of assessing general trade-off between accuracy and resources, these are often useful in the characterization of single-purpose approximate components. However, their relevance to real-world applications is often limited. Addressing these resource-quality requirements, only ApproxFPGAs models the resources on FPGAs and uses structural similarity measure (SSIM) as quality metrics for an image filter application. However, their approach considers only arithmetic components, and the area model accounts for only the number of LUTs. The information on how to handle the resources for different approximations, which use dedicated DSPs or BRAMs, is not specified in their approach. Similarly, GRATER, proposed for FPGAs, uses OpenCL kernel transformation for integrating the approximations, and this approach considers only multi-level precision scaling as an approximation technique. Therefore, this approach cannot extensively exploit the resource benefit of approximate computing on FPGA.

Addressing the above requirement for maximizing approximate computing benefits, we propose AxCGA, which differs from previous works in several ways. Using AxCGA, a designer can select and combine appropriate single-purpose approximation techniques across different abstraction layers and globally optimize tunable parameters exposed by each technique in conjunction using multi-objective optimization (MOO). We adapted GA, which globally explores highly complex design spaces with a reasonable number of fitness evaluations and non-dominated sorting selection, NSGA-II [225] to obtain Pareto-optimal or near Pareto-optimal solutions. Due to the global optimization approach, AxCGA implicitly considers interactions and error propagation between different system components during the optimization. Similarly, NSGA-II can inherently maintain the convergence and diversity in the obtained Pareto solutions. Therefore, an effective trade-off analysis is feasible for a designer to exploit the maximum approximation benefit with desired quality level. In contrast to of other GA-based approaches which often require time-consuming hyperparameter optimization, we proposed an adaptive GA approach that dynamically provides hyperparameters during the DSE. This adaptive approach avoids repeating the DSE experiments to fine-tune the hyperparameters for each target application and make the AxCGA generic. Additionally, AxCGA directly targets FPGA designs using established design flows. Our approach uses a library of reusable approximate components whose resources are estimated at the beginning as a one-time effort. This library includes resource models for each component which take into account the specific FPGA architectural characteristics, including the use of dedicated DSP units, BRAMs, and adaptive LUTs. During the DSE, AxCGA estimates the total resource utilization as a sum of each of these resources from all the system components, and the total power consumption is predicted directly from these estimated resources. These simple yet accurate models estimate the resource utilization during the DSE without any time-consuming iterative synthesis of parameter configurations for each target application. Furthermore, multiple established quality metrics are available in the framework, and designers can choose desired quality metrics, allowing them to rely on application-specific metrics.

To demonstrate the uses of AxCGA, we consider two image processing case studies, such as an RGB to $YCbCr$ color space conversion and a display rendering pipeline typically used in digital cameras to adapt images for display on a specific monitor. The DSE experimental results on these applications show that the AxCGA is able to construct a suitable Pareto front with respect to the quality-resource trade-off. We additionally employed autoAx DSE on these two case studies for comparing the performance of AxCGA. Since the modeling part is not a contribution of this thesis, we use the same resource and quality models in both the AxCGA and autoAx DSE experiments, and the comparison primarily considers the performance of the exploration strategy in determining the quality-resource trade-off. The performance of each DSE approach is analyzed individually based on the hypervolume indicator, which reflects both convergence and diversity of evolved Pareto-optimal solutions. The performance comparison shows that AxCGA achieves similar performance to autoAx with only around 5% and 25% of the computational effort, respectively, in the above applications. Finally, in order to estimate the actual savings on an FPGA, we selected a number of parameter configurations from the Pareto fronts and performed synthesis & fitting using standard FPGA design flow.

The remaining sections in this chapter are organized as follows. Section 3.2 formulates the DSE problem, and Section 3.3 gives a high-level overview of our proposed AxCGA methodology. Thereafter, Section 3.5 describes the modeling approach for resource, power, and quality. Then, the GA-based DSE is explained in detail in Section 3.6. The Section 3.7 explains the two case studies considered in this work, such as RGB to $YCbCr$ conversion and display rendering pipeline, and discusses the DSE experiments and results from approximations on these applications. The Section 3.8 compares performance of adaptive and non-adaptive version of AxCGA, and Section 3.9 compares AxCGA with the state-of-the-art autoAx DSE. Finally, Section 3.10 summarizes our findings and concludes this chapter.

3.2 Problem Formulation

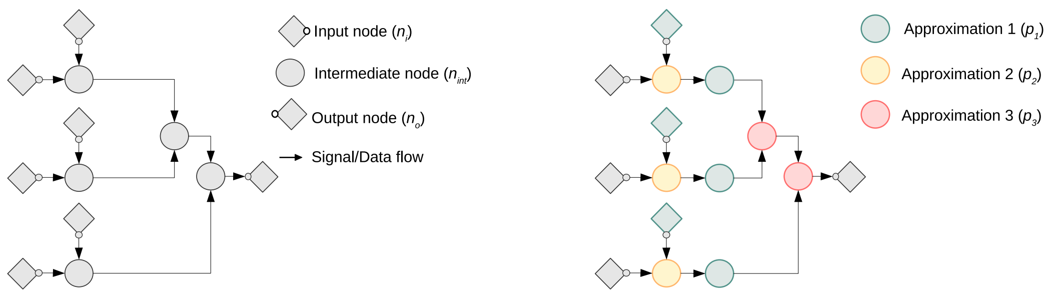
The goal of the AxCGA is to determine the quality-resource trade-off in an FPGA application when multiple approximate computing techniques are employed. In AxCGA, we use a DFG-based approach to represent signal flow in an application. The signal flow in a target application N is represented as a DFG in which each node $n \in N$ constitutes a typical signal operation or an input or output signal. Figure 3.1a shows an exemplary DFG that includes different input nodes n_i, an output node n_o, and multiple intermediate nodes n_{int}, and these nodes are connected using directed signal or data flow. In general, DFG of an application N is defined as:

$$N \in \{n_{i\{1,..,a\}}, n_{int\{1,..,b\}}, n_{o\{1,..,c\}}\}, \tag{3.1}$$

where a, b, c represents the respective number of each node type. With an application DFG, a designer can annotate nodes that can be approximated and specify suitable state-of-the-art approximation techniques, if applicable. Additionally, the designer can add specific nodes or approximations such as precision scaling to the directed signal flow as shown in Figure 3.1b. Such a modified DFG with user specifications is named as *annotated DFG*. In general, nodes in the annotated DFG exposes a set of parameters $p \in P$, which are used to a) select among different variants of state-of-the-art approximation techniques, if applicable, and b) tune the strength of the approximation. Therefore, the parameter set P is defined as:

$$P \in \{p_{\{1_1,..,1_y\},..,\{x_1,..,x_y\}}\}, \tag{3.2}$$

where x is the number of nodes considered for the approximation, and y represents the number of approximation parameters considered for each annotated node. In the annotated DFG example 3.1b, three different approximation types are specified. Even though the same approximations are used to annotate multiple nodes, their designer-specified parameters and their ranges might be different. In general, an annotated DFG includes a list of parameters P that configure all the annotated nodes in the target application N. Therefore, together with these parameters, an annotated DFG represents the design space, and the DFG with a typical parameter configuration of an application is denoted as a *candidate DFG*.



(a) Data flow graph (DFG)

(b) Annotated data flow graph

Figure 3.1: An exemplary application with a number of intermediate nodes represents specific operations

In AxCGA, the trade-off estimation is formulated as a MOO problem since the target is to identify the Pareto-optimal points that trade in application quality for the resource benefits. Because of possible interactions and error propagation between the nodes in a DFG, an effective trade-off analysis is not feasible by optimization of individual nodes with their possible configurations. Therefore, a global DSE is necessary, considering the combined parameter sets from all the annotated nodes. However, the design space complexity increases exponentially with the number of annotated nodes and their exposed parameters in the case of a global optimization approach. The overall objective of this work is to identify the Pareto trade-off for a target FPGA-based application N, which is represented with an annotated DFG with P possible parameters.

3.3 Methodology Overview

The proposed AxCGA addresses the problem of resource-quality trade-off estimation in an approximate computing application. An overview AxCGA design flow is depicted in Figure 3.2.

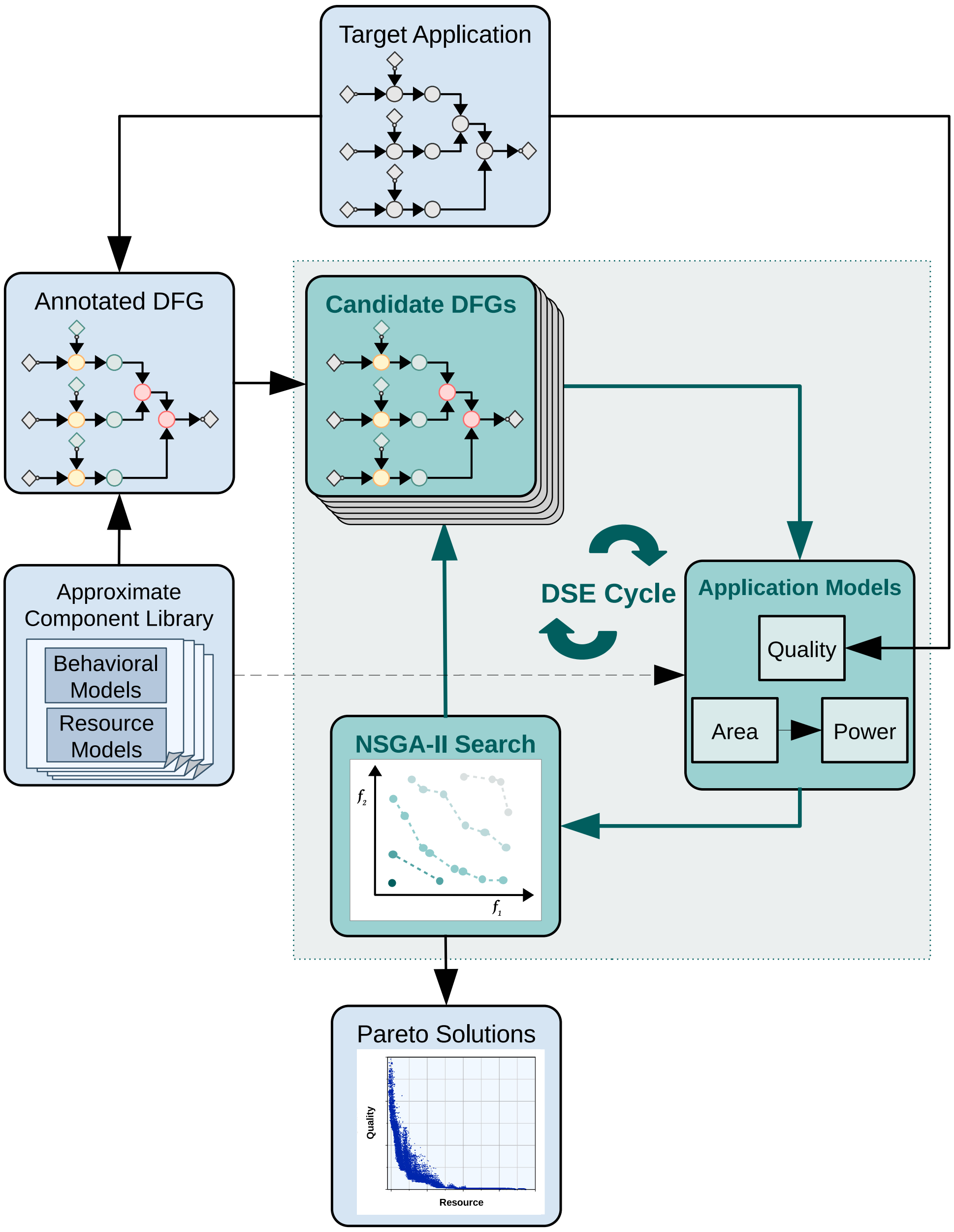


Figure 3.2: Overview of AxCGA design flow

To facilitate DSE of complex FPGA applications, we employed a divide and conquer approach in which an application is represented as a directed DFG with nodes representing different components or operations in the application. Therefore, a designer first creates an application DFG by specifying the input and output signals of the application as input nodes and output nodes. In addition, successive operations on the input signals are represented as different intermediate nodes, and these nodes are connected each other using directed data flows. Different parts of the AxCGA framework are implemented in Python using various open-source libraries, and standard graphic libraries such as `NetworkX` [214] can be used to create DFG of an application.

While creating the DFG, a designer can specify one or more suitable approximation techniques and their parameters that can be added to the directed data flows or replace the nodes as described in Section 3.2. These approximations are selected from a predefined library that contains multiple variants of approximate components for different signal or image processing operations. The resulting annotated DFG uses an instance of an approximate component class taken from the approximate component library to define the functionality of the annotated nodes. To estimate the resource utilization during the DSE, each library component includes a resource model that defines resource utilization as a function of its configuration parameters. Additionally, each component includes a behavioral model that performs a bit-accurate software simulation for quality estimation.

In an annotated DFG, a typical annotated node can be switched between an accurate and one among the specified approximate component class using their object properties. Additionally, each approximate component class can be parametrized using its object properties. With multiple approximation choices and their parameters for each node, the annotated DFG produces numerous candidate DFGs. Therefore, identifying the candidate DFGs corresponding to parameter combinations that effectively trade-off application quality for resource benefits is challenging. Addressing this issue, AxCGA performs an efficient GA-based DSE search and identifies the Pareto-optimal candidate DFGs or solutions.

During a DSE, different candidates DFG have to be evaluated for their fitness values. Therefore, fast and accurate fitness estimation is required for an efficient and effective search. Since each candidate DFG is a typical parameter configuration of an application, these fitness values must be estimated on an application level. An application model should provide fitness values as resources, such as area or power and quality values specific to an application. For the area estimation, we employ a simple divide-and-conquer approach. Initially, the area of each node in a typical candidate DFG is estimated, and such values from all the nodes in a candidate DFG are summed up for an overall area estimation. The power consumption of the candidate DFG is estimated directly from this total area. For each annotated node, this information is taken from the resource models corresponding to each approximate component. However, for quality estimation, a divide-and-conquer approach is not feasible due to error propagation between the components, whereas a global quality estimation is necessary. The quality models use behavioral models associated with each component node which can simulate both approximate and reference versions of a specific node. The characterization of different approximation techniques and modeling on component and application levels are already published in [216] and not a contribution of this thesis. However, the library of approximate components used in AxCGA is explained in the following section, and the core idea of application-level modeling is described in Section 3.5 to guide a reader through the AxCGA methodology.

In addition to fast models, a clever search technique is also required for an efficient and effective DSE. In AxCGA, we employed a GA-based approach, which can find optimal parameter configurations with a few fitness evaluations. We encode parameter configurations with real value genes considering forward and backward parameter dependencies to avoid non-useful fitness evaluations. Additionally, the inherent convergence and diversity mechanism in NSGA-II finds well-distributed Pareto-optimal solutions after a few search iterations. To avoid application-specific GA hyperparameter optimization, an adaptive approach is proposed that supplies the hyperparameter values on the fly during the DSE. The different phases of GA-based DSE approach is explained detailed in Section 3.6. Overall, the formation of candidate DFGs, fitness estimation, and DSE search together forms an iterative cycle in AxCGA.

3.4 Approximate Component Library

We selected different state-of-the-art approximate computing techniques to form a library of approximate components in AxCGA. Each component in the library contains three major elements: a parametrizable hardware implementation, a behavioral model, and a resource model. A hardware implementation of a specific component is provided as a VHDL entity that is parametrizable by varying the I/O width and configuring the approximation via generics. Multiple approximation variants of a specific operation often exist in the library for an intermediate node. For example, an addition operation can be replaced with multiple types of approximate adders. In such cases, a top-level entity of this node is a generic wrapper that instantiates the selected components in VHDL. However, VHDL generics are used to control the width of the internal signals for methods like precision scaling, which applies to a directed signal flow in a DFG.

Both the component and behavioral models are implemented as Python classes. A `process()` method can access the behavioral model, which provides a bit-accurate simulation result, and a `report_area()` can fetch the resource model, which supplies the consumed FPGA resources. Such resource models can be formed either with a one-time effort using an analytical or machine-learning approach or directly taken from published libraries like in [215]. Table 3.1 shows the approximation methods selected and implemented in the AxCGA approximate component library. These methods are altered for better parametrization, for example, configurable input bitwidths for arithmetic components. The characterization and modeling approach of these methods are detailed in [216]. A designer can easily append more approximation component classes to this library with their models if desired.

3.5 Application Models

Each candidate DFG with a typical parameter configuration represents an application with a certain degree of approximation. Such candidate DFGs are characterized by their fitness values in terms of resource usage and application quality. The classical hardware description language (HDL) flow requires time-consuming synthesis, fitting, and place & route operation to estimate

Table 3.1: Overview of approximation methods available in the component library. Some of the methods are modified from the original proposal for better parametrization.

Technique	**Methods**	**Configurable Parameters**
	Circuit Level	
Adder	Accurate	Width
	LSA [132]	Width, Split, Input Select
	Median Adder [140]	Width, Split
	LOA [141]	Width, Split
	Sloppy Adder [142]	Width, Split
	OLOCA [143]	Width, Split
	HOAANED [144]	Width, Split
Multiplier	Accurate	Width
	BAM [141]	Width, HBL, VBL
	DRUM [169]	Width, Core Size
	RoBA [175]	Width, Operation Mode
	RBA [177]	Width, Truncation Width
	LM [179]	Width, Truncation Width
	Algorithm/Software level	
Table Based Method	Hierarchical segmentation [67]	No. of Section and Sub-segments
Precision Scaling	Adapting signal widths [14]	Width

the resource usage of an FPGA design. Similarly, quality estimation using gate-level simulations of such a design is also inefficient. Therefore, fast estimation of these fitness values is necessary to perform an efficient DSE that involves an iterative search heuristic. The following sections explain the resource and quality models used to estimate the fitness of candidate DFGs in AxCGA.

3.5.1 Resource Models

In our AxCGA, the resource consumption of a candidate DFG with different nodes is estimated using a divide-and-conquer approach. Figure 3.3 shows an overview of a two-step application-level resource modeling approach. The total area consumed by a specific design configuration is initially estimated from resources corresponding to individual nodes in a candidate DFG. After that, the total power consumption is estimated from this area using an approach similar to HAPE [217]. The following subsections explain each step in resource modeling in detail.

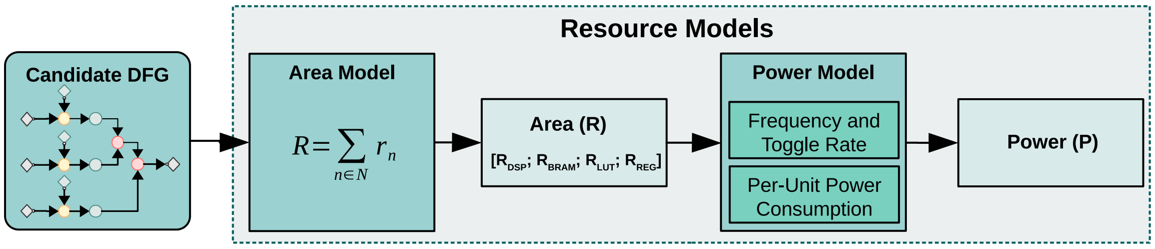


Figure 3.3: Two-step application-level resource modeling

3.5.1.1 Area Model

Contemporary FPGAs have different types of resources available, such as LUTs, registers, DSP units and BRAMs to implement complex digital image or signal processing applications. Depending on the characteristics of an application and type of available resources on a target FPGA, a specific design uses one or many such resources to implement different operations. For example, arithmetic operations such as multiply or multiply-accumulate can be directly implemented using DSPs. However, due to the customizable logic of LUTs, these same functionalities can be implemented using LUTs as well. Similarly, both the BRAMs and the registers can be used to store data on an FPGA. However, BRAMs often stores large data, whereas resisters are often used to store intermediate signals. Due to these architectural characteristics of FPGAs, the area of a design cannot be estimated as a single chip area value. Therefore, the area model estimates a vector R, representing the total number of each FPGA resources consumed by a design configuration.

$$R = [R_{DSP};\ R_{BRAM};\ R_{LUT};\ R_{REG}], \tag{3.3}$$

Since a typical node in a candidate DFG represents a component or operation, the resources consumed by such nodes are estimated individually first using the resource models provided with the approximate component in the library. The resource consumed by any non-approximated nodes in an application DFG can be estimated from a one-time synthesis of the design and added to corresponding resource of all candidate DFGs evaluated during the DSE. However, for any DSE problem, the fidelity, which represents the degree to which the trend across a model estimation is similar to traditional HDL resource estimation, is more important than the accuracy of models. Therefore, constant resource numbers from non-approximated components can be safely ignored in the resource estimation. For each component in the library, a resource model offers a function to calculate the resource consumption r_n based on possible approximation parameter p_n.

$$r_n = f(p_n). \tag{3.4}$$

where r represents each resource types on an FPGA.

$$r = [r_{DSP};\ r_{BRAM};\ r_{LUT};\ r_{REG}]. \tag{3.5}$$

A function call to the `report_area()` of each component class provides these numbers corresponding to individual resources. Finally, the overall system resources can be expressed as a sum of resources from all individual component nodes $n \in N$.

$$R = \sum_{n \in N} r_n, \tag{3.6}$$

Such a divide-and-conquer approach facilitates an instantaneous area estimation of candidate DFGs during a DSE.

3.5.1.2 Power Model

A fast power estimation is necessary to perform an efficient DSE in complex applications with multiple approximation parameters. Therefore, we adopted a power model similar to HAPE proposed by Makni et al. [217] that uses analytical models for a high-level power estimation without any RTL implementation.

Power consumption on any semiconductor devices, including FPGAs, consists of two main parts: static and dynamic power. Therefore, the total power P is a sum of both the static and dynamic power of all resources a design consume. The static power corresponds to the power dissipated by underlying transistors when they are not switching, whereas the dynamic power depends on the switching activity of transistors over time. The core idea of our power modeling approach is to identify the per-unit power consumption for each FPGA resource type as the static and dynamic power and then store these values in the model in advance. Thereafter, the total power is estimated during the model evaluation as the sum of these two components from all the different resources used in a design. For a specific candidate DFG, the static power P_{static} of a typical resource type $t \in r$ is estimated as a product of the number of such resource type from the area model R and the per-unit static power Q_{static} of the specific resource type.

$$P_{static,t} = R_t * Q_{static,t} \tag{3.7}$$

Due to the dependency on underlying transistor switching, dynamic power estimation considers both toggle rate α and clock frequency f_{clk}. The switching activity is estimated as a number of signal transitions per seconds $f_{clk} * \alpha$. Therefore, the dynamic power $P_{dynamic}$ of each resource type $t \in r$ is estimated as:

$$P_{dynamic,t} = R_t * Q_{dynamic,t} * f_{clk} * \alpha, \tag{3.8}$$

where $Q_{dynamic}$ is the per-unit dynamic power of the specific resource type t. The clock frequency f_{clk} depends on the design target and can be directly specified by a designer. However, the toggle rate α estimation depends on different factors such as the target application, employed approximation techniques, and the data it processes, and an accurate estimation of this value for each possible parameter configuration is time-consuming. Therefore, we estimate the toggle rate as an average bit-level switching of all internal signals using behavioral simulation of the reference DFG with an input dataset.

Finally, the total power P is estimated as a sum of static and dynamic power from all resource units corresponding to each FPGA resource type.

$$P = \sum_{t \in r} (P_{static,t} + P_{dynamic,t}). \tag{3.9}$$

The per-unit power consumption Q_{static} and $Q_{dynamic}$ can be obtained for different FPGA resources $t \in r$ either using a generic method or with an application-specific method. In the generic method, we estimate these values from specific vendor tools such as Early Power Estimator (EPE) [218] for Intel FPGAs or Xilinx Power Estimator (XPE) [219] for Xilinx FPGAs. These tools are spreadsheet-based and allow for estimating the power consumption of an FPGA design based on the number of instantiated resource units. We can directly set the parameters such as the name of target FPGA and operating temperature based on the design target. However, for the unknown routing factor, we used a standard value at this stage.

In the second application-specific method, we include the routing effect also to improve the model accuracy further. Initially, a few random approximation parameter configurations and the reference design are synthesized and performed place & route. Thereafter, the power reports are generated using tools such as Power Analyzer from Intel [220]. By averaging the power consumption per number of resource units consumed for each resource type across all the configurations, the per-unit power consumption can be estimated in terms of both Q_{static} and $Q_{dynamic}$. Our test and verification of both methods across the 100 random configurations show that the application-specific method is more accurate than the generic model. However, the fidelity is comparable in both methods, which is essential for a DSE.

The scope of this dissertation is limited to the approximation of pixel streaming applications. Due to the simplicity of our power models, which estimate power from the consumed FPGA resources, the current version of AxCGA supports only such streaming applications without any retrospective communication between DFG nodes. However, with appropriate resource models, the AxCGA methodology can be extended in future to various applications.

3.5.2 Quality Model

In approximate computing applications, appropriate quality metrics are important to provide reliable quality-thresholds and make suitable design decisions. A quality model should be fast, accurate, and interpretable for a designer. In many state-of-the-art approximate computing research, quality is often measured using basic metrics such as mean squared error (MSE), absolute error, or Hamming distance. However, the relevance of these metrics to real-world applications is often limited in making valid design decisions.

In AxCGA, we provide different state-of-the-art reference metrics such as PSNR, CIELAB ΔE [221], or SSIM [222], together with other basic metrics such MSE, mean and maximum absolute error for the quality estimation. Similarly, other quality metrics can also be integrated easily into the framework. The overall idea is that designer can choose a quality metric depending on the individual needs of an application. For example, a color processing application whose quality is subjectively evaluated with human eyes requires metrics such as CIELAB ΔE, which measure color difference in a perceptually uniform color space.

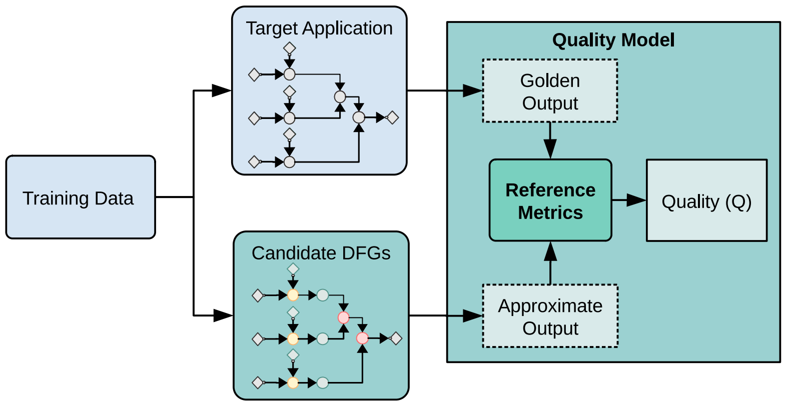


Figure 3.4: Overview of global quality modeling approach using reference metrics

Due to the error propagation through intermediate nodes of a design, quality metrics are globally estimated from the output nodes of a DFG. Figure 3.4 shows an overview of the AxCGA quality modeling approach. First, a suitable image dataset is processed with a reference system DFG without any approximations, and the resulting golden output images are stored in the quality model. Then, during the DSE, each candidate DFG is processed with the same image dataset to estimate the quality of individual approximation configurations. The golden output and the approximate output from a candidate DFG are used by the selected metric to estimate a quality value. A `process()` function call on the application level can provide a bit-accurate software simulation of a DFG, and thereby, the quality values of each candidate DFG can be estimated on the fly during the DSE.

3.6 Design Space Exploration

An approximation technique in an application exposes different parameters to configure the strength of approximations and introduce errors that often propagate over successive application components. Due to this error propagation, each component cannot be optimized in isolation, necessitating a global DSE approach on a joint parametrization of all the approximate components in an application. However, combining multiple such parametrizable methods on different abstraction levels exponentially increases potential parameter configurations in the design space. For example, a simple approximate 16-bit two-input adder with input bitwidth varying between $1-16$ can have $16^2 = 256$ different parameter combinations. Combining this with a subsequent precision scaling, where the adder output bitwidth can vary between $1-16$, ideally, exposes $256 * 16 = 4096$ different parameter configurations. A typical configuration has to be evaluated for fitness values using the quality and resource models to estimate its approximation potential. Due to the extensive time for fitness evaluation in a complex application with multiple such approximation methods, a brute-force search is realistically impossible. This demands an intelligent DSE that searches highly complex design spaces with fewer fitness evaluations to determine the quality-resource trade-offs.

Even though with precision scaling, the bitwidth of the adder output can vary between 1 – 16, the range of bitwidth, where precision scaling has an influence, depends on the actual output bitwidth of the first adder. Due to this dependency, valid parameter configurations are only 2600 from these two approximations. Therefore, the DSE has to consider potential interactions between different approximation parameters to avoid irrelevant fitness evaluations.

Addressing these challenges, we use a global GA-based DSE in AxCGA to efficiently explore the complex design spaces. We used this DSE methodology in our different publications from the joint project [23, 24, 25, 27, 28, 29]. However, this thesis presents a refined version of the DSE with updated genetic operations and an adaptive hyperparameter setting technique to avoid application-specific hyperparameter tuning. The subsequent sections define approximate computing as a MOO problem and explain how our GA-based DSE solves this MOO problem in detail.

3.6.1 Approximate Computing as Multi-Objective Optimization (MOO)

Approximate computing improves resource utilization in an application, often at the expense of application quality. Therefore, the quality and resource objectives invariably conflict in the context of approximate computing, which makes approximate computing a multi-objective optimization problem by definition. Many state-of-the-art techniques listed in Table 2.1 address this problem using single-objective optimization techniques by transforming the multiple objectives into an aggregated single-cost function. In addition, specific weights for each objective are often used in this transformation to incorporate a bias toward search directions in a design space. However, defining these weights is a non-trivial problem and restricts the potential of a DSE in finding globally optimal solutions. Nevertheless, a multi-objective optimization approach excludes the difficulty of defining weights to form an aggregated objective value. Furthermore, a suitable multi-objective optimization can efficiently and effectively determine the trade-off between conflicting design objectives. An in-depth analysis of approximate computing as a multi-objective optimization is often missing in state-of-the-art works. This thesis fills the gap between approximate computing techniques and quality-resource trade-off analysis by proposing a suitable GA-based multi-objective DSE to effectively explore the design space and determine this trade-off in approximate computing applications.

An approximate computing problem can be defined as minimizing or maximizing specific objectives and determining Pareto-optimal solutions that show the trade-off between these objectives. In AxCGA, GA-based DSE is implemented with the support of the DEAP framework [223], and AxCGA intrinsically treats all the applications as multi-objective minimization problems. The hardware resources such as power or area must be minimized. In contrast, application quality can either be estimated directly, for example as PSNR, where a maximization is necessary or be evaluated as error values such as MSE or CIELAB ΔE where a minimization is essential. A designer can configure various DSE options using `GaOptions` and `GaSelection` class in AxCGA. During the instantiation of `GaSelection`, the designer can optionally specify weights for each objective irrespective of default weight -1 for minimization. Therefore, a weight of +1 treats the corresponding target objective as maximization during the DSE in AxCGA.

In general, approximate computing as an m objectives minimization problem is formulated as:

$$\begin{aligned}&\text{minimize:} f(\mathrm{x}) = (f_1(\mathrm{x}), f_2(\mathrm{x}), \ldots, f_m(\mathrm{x}))^T,\\ &\text{subject to } \mathrm{x} \in S,\end{aligned} \tag{3.10}$$

where $x = (x_1, x_2, ..., x_n)^T$ is a vector of optimization variables named decision vector, $z = f(\mathrm{x}) \subset \mathbb{R}^m$ is an objective vector, and $S \subset \mathbb{R}^n$ is an n-dimensional design space. To identify a Pareto front in such a MOO problem, Goldberg originally suggested the use of non-domination ranking and selection in 1989 [224], and later in 2001, Deb et al. proposed the well-known NSGA-II selection [225] which is the core of our GA-based DSE in AxCGA.

3.6.2 GA-based DSE Approach

GA is an evolutionary metaheuristic optimization approach inspired by Charles Darwin's natural evolution [226]. Genetic operations such as mutation and crossover, together with suitable selection methods, mimic the survival of the fittest in the evolution theory. Transferring this to optimization problems, the GA can evolve optimal solutions by selecting the fittest decision vectors over generations of an iterative heuristic. Therefore, defining appropriate genetic operations and a selection method is essential for successful optimization. GA is distinguishable and beneficial from other optimization approaches in complex optimization problems. GA with non-dominated selection methods consider multiple competing objectives simultaneously during the optimization to form Pareto-optimal solutions. The non-dominated selection methods such as NSGA-II inherently maintain convergence and diversity in selected solutions from a generation for a highly explorative and exploitive search. In general, evolutionary algorithms can be used to optimize problems with various Pareto shapes, such as convex, concave, continuous, or disconnected [227]. Therefore, GA can handle optimization problems with complex Pareto front shapes. A typical decision vector represented in GA is called an individual or a chromosome, and GA is a population-based approach where many independent individuals are evaluated for their fitness at the same time. Therefore, a high parallelization of computation-intensive fitness evaluations is feasible on multiple computing threads during the optimization, depending on the available computational resources. This increases the optimization speed approximately by a factor of available computational cores for fitness estimation. In contrast, a reasonable number of such fitness evaluations is a bottleneck for optimization techniques commonly used in state-of-the-art approximate computing, such as hill climbing, tabu search, or greedy search, which iteratively evaluate a single individual at a time. Additionally, GA is a global optimization approach that aims to identify globally optimal solutions in a complex optimization problem. Instead, local search methods ideally find the globally optimal solution if there are no local optima existing. Therefore, a local search technique has a higher chance to stuck on local optima in complex problems with multiple local optima. Other well-known population-based optimization techniques, such as particle swarm optimization (PSO) [228] or simulated annealing (SA) [229] can also be used for the MOO. However, these methods need additional mechanisms for improved diversity in identified solutions. In general, GA with NSGA-II algorithm is relatively simple and easy to implement, can handle many decision variables and multiple conflicting objectives, and can explore Pareto-optimal solutions by maintaining both the convergence and diversity at the same time. In addition, elitism in NSGA-II helps to maintain good solutions throughout the optimization generations. Even though other metaheuristics or problem-specific optimizations might perform well in some problems, the above properties make GA and NSGA-II popular in many real-world optimization problems.

3.6.3 AxCGA DSE Overview

An overview of GA-based DSE in AxCGA is shown in Figure 3.5. The design flow starts with AxCGA initialization, where both application-specific and GA-specific initialization are performed. As a result, an initial population, a set of individuals representing candidate DFGs with different parameter configurations is formed. In the first DSE generation, the initial population undergoes genetic operation such as crossover and mutation and produces new individuals, named offspring, by inheriting the genetic properties of their parents. The individuals in the initial population serve as the parents in the first generation. After that, the application models estimate the fitness values of each offspring in terms of area, power, and quality. Since the fitness of the initial population is unknown in the first generation, initial population is also evaluated for their fitness. Subsequently, a non-dominated sorting selection is performed on a combined set of parents and offspring based on their fitness values to determine the fittest individuals in the first generation. As a result, a subset of individuals is selected, serving as parent population of the second DSE generation. A Pareto front is created using the fitness values of this non-dominated subset, which will be iteratively updated over the DSE generations. Thereafter, the DSE checks whether the desired stop condition is met, which often depends on how good is the evolving Pareto front. Closing the cycle, the selected parent population undergoes genetic operation if the stop condition does not meet. This will result in a new set of offspring, and the DSE cycle repeats until the stop condition is satisfied. Finally, a set of evolved Pareto-optimal configurations are returned, which trade-off the target design objectives. We use well-known $(\mu + \lambda)$ evolution strategy [230], where μ number of offspring are evolved in a generation from λ number of individuals selected from the previous generation. Each of these DSE phases is explained in detail below.

3.6.3.1 AxCGA Initialization

A designer can define an application, selects appropriate approximation techniques from the component library, and specify the DSE objectives in the initialization phase. In general, different initializations in AxCGA can be broadly categorized into an application-specific and a GA specific initialization.

Application-Specific Initialization

In the application-specific initialization, a designer has to create an application DFG first, as explained in Section 3.2. The input and output bitwidth of the target application, relevant training data for DSE, and intermediate node related inputs also have to be specified while creating the DFG. Subsequently, annotated DFG is created to incorporate the relevant approximate computing techniques. The specified bitwidth of both the input and output often defines the parameter ranges of many of these approximate computing techniques. This initialization phase also includes

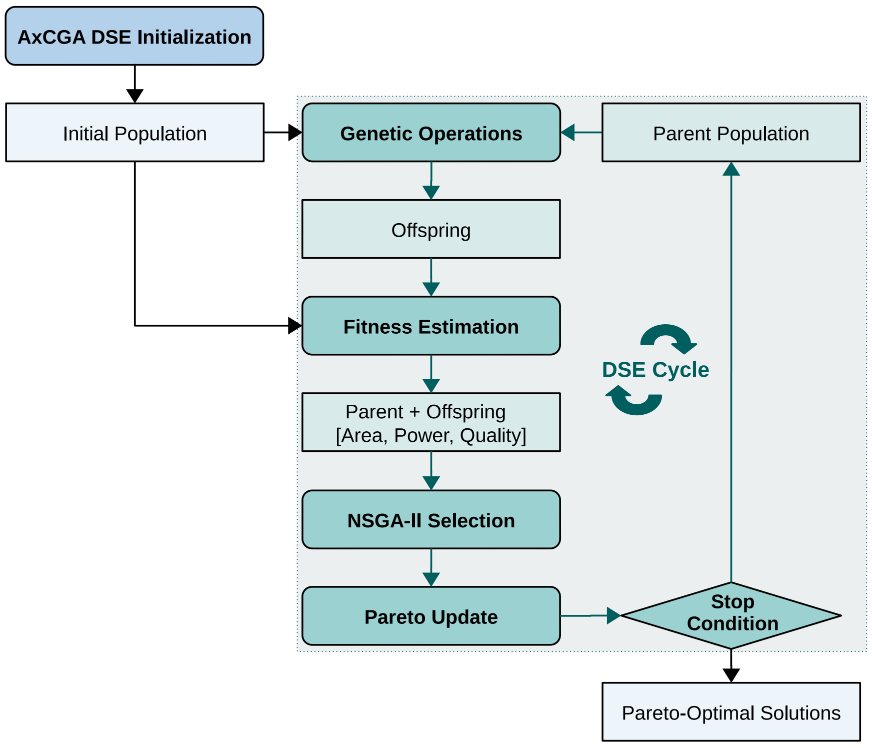


Figure 3.5: GA-based DSE in AxCGA

specifying a desired reference metric in the quality model and a one-time estimation of the per-unit power consumption of each resource in the case of an application-dependent power model and a global toggle rate from the input training data.

GA-Specific Initialization

GA specific initialization primarily includes defining the DSE objective function, encoding the annotated application DFG into genes or chromosomes, formation of the initial population, and setting up the GA specific hyperparameters. Additionally, a Pareto front class is initialized, which will be iteratively updated over the DSE generations. The following sections describe each GA-specific initialization in detail.

Objective Function: The first step in the GA specific initialization is defining the objective function in a DSE. The output quality is one of the DSE objectives that must always be optimized due to the nature of approximation computing applications. Multiple quality measures can be chosen for a single application to ensure a valid design decision in an application. In an example pixel processing application, reference metrics like CIELAB ΔE can be expressed as both the mean and maximum values, where the mean value provides the quality estimate over an entire training image dataset, whereas the maximum error gives the worst-case error from individual pixels. The resources can be specified as either power or individual area resources. However, this thesis is limited to pixel streaming applications where power can be directly derived from the area components. Therefore, the case studies considered in this dissertation use power as the resource objective.

GA Encoding: A candidate DFG corresponding to a typical parameter configuration needs to be encoded into a chromosome or individual to perform genetic operations in GA. Some of the intermediate nodes in an application DFG may be interdependent, and the genetic operations need to consider these dependencies in addition to other application constraints. The classical binary encoding of parameters into a gene is often not useful in real-world approximate computing applications. Therefore, the parameters of an application DFG are encoded as a nested list of real-value genes in AxCGA.

An application may contain different processing *stages* with specific functionalities, and each stage is treated independently, considering its characteristics for the genetic operations. For example, among the two case studies presented in this thesis, the first RGB to $YCbCr$ color space conversion is a single-stage application, whereas the second case study display rendering pipeline contains three different processing stages, including a color space conversion stage. Therefore, with a nested list of real-value gene encoding, the first hierarchical level groups the parameters from each application stage if more than one stage exists. Within each application stage, a group of interdependent nodes or a specific intermediate node without any dependency represents a *module*. Therefore, such modules within a stage are independent of each other and are grouped in the second hierarchical level. In case of a module with a group of interdependent nodes, a third-level sublist groups these dependent nodes in each module. Finally, a fourth-level sublist clusters the parameters exposed by each node within a module that may influence the parameter of another node in the same module.

A real-value encoding is illustrated with an example annotated DFG with three processing stages, each containing three independent modules. Initially, these three application stages are grouped to form the first-level list in an individual or chromosome.

$$\text{Individual } (I) = [P^{Stage\ 1}, P^{Stage\ 2}, P^{Stage\ 3}] \tag{3.11}$$

Similarly, the second-level sub-list groups the modules from each stage, further represented as

$$P^{Stage} = [P^{Module\ 1}, P^{Module\ 2}, P^{Module\ 3}] \tag{3.12}$$

Each module contains nodes A, B, C, where the node parameters depend on each other and have x, y, z number of independent instances within a module. Therefore, the third-level sub-list cluster such nodes by maintaining the dependency.

$$P^{Module} = [P^A, P^B, P^C] \tag{3.13}$$

Finally, the fourth-level list groups the parameters p exposed by independent instances of these nodes within the same module.

$$P^{Module} = [[p_1^A, .., p_x^A], [p_1^B, .., p_y^B], [p_1^C, .., p_z^C]] \tag{3.14}$$

Each node parameter p defines either an approximation variant, for example, the type of approximate adders in case of an adder component, or the strength of a specific approximation, for example, the bitwidth of a signal in precision scaling. While grouping the interdependent nodes within a module, the priority order in this dependency propagates over the DFG nodes needs to be considered. An example module with two dependent nodes, such as an approximate adder followed by a precision scaling, is shown in Figure 3.6, illustrating the direction in which the parameters propagate. The precision scaling node has a lower order *Priority*-1 among the two nodes since its parameter ranges depend on the adder node with higher *Priority*-0, where *Priority*-0 > *Priority*-1. Therefore, during the real-value encoding, parameters of the *Priority*-0 node encode first, and thereafter, the node with lower *Priority*-1 encodes by constraining its parameters with the previous higher order node.

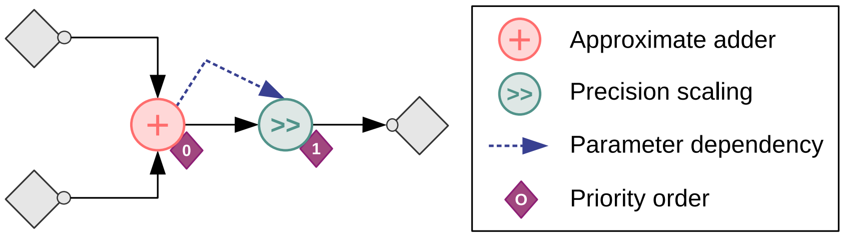


Figure 3.6: An example DFG module with approximate adder followed by precision scaling

In general, for large DFG modules with multiple such dependent nodes, the parameters of a lower-priority node are defined by the output of the nodes with one higher-level priority. Therefore, the priority order of parameter propagation within the modules in Equation 3.13 is

$$Priority\ p^A > Priority\ p^B > Priority\ p^C. \tag{3.15}$$

A typical hierarchical level in this real-value encoding scheme vanishes if the level contains only a single entry. Other examples of hierarchical sub-list-based real-value encoding can be seen in Section 3.7.1.4 and Section 3.7.2.4 from the case studies considered in this thesis.

Initial Population: The initial population in AxCGA is formed by selecting random parameters p for each node n in an annotated DFG. While choosing parameters p, the interdependence within a module according to the priority order has to follow. Additionally, each parameter has to meet its user-defined constraints and the constraints added from a higher-priority node. The initial population act as the parent population in the first DSE iteration.

Designers usually have a certain quality-threshold in an application and analyze trade-offs within a region defined between maximum quality and this quality-threshold for suitable design decisions. The quality-threshold value varies with a specific target quality level in an application. However, maximum quality can be obtained from a reference design by selecting the approximation parameters with zero approximation strength. In AxCGA, one individual within the initial population can be optionally created with parameters corresponding to the reference design. If more than one approximation variant exists for any node, a random type with zero approximation strength is selected for the reference design. By seeding such a reference individual into the initial population, the AxCGA biases the GA-based DSE to explore a region near to the maximum quality, which often might be interesting to a designer.

Hyperparameter Settings: In a classical GA approach, specific hyperparameters such as the number of individuals in a parent population μ, the number of offspring generated from the parent population λ, probabilities of genetic operation such as crossover probabilities P_c and mutation probabilities P_m are supplied by the designer. However, finding optimal hyperparameters would be time-consuming and need to repeat DSE multiple times for every target application. To avoid this hyperparameter optimization and make AxCGA generic to different applications, we used commonly used hyperparameter values in our previously published works [23, 24, 25, 27, 28, 29]. These parameters are provided during the instantiation of `GaOptions` class in AxCGA. To further improve the performance and generality of AxCGA in distinct approximate computing applications, the current version of AxCGA uses a reinforcement-based self-adaptive approach that provides hyperparameters dynamically during the DSE (cp. Section 3.6.4). A designer can set a flag `adaptive = true` for using the adaptive version in AxCGA. Finally, the parameters determining the stop condition of the DSE are also specified during the GA initialization.

3.6.3.2 Fitness Estimation

The fitness estimation phase in AxCGA design flow estimates the fitness values in terms of initialized DSE objectives using application models explained in Section 3.5. In the first DSE generation, the fitness values are estimated for individuals in both the initial and offspring populations. From the second generation onwards, the parent population is a subset selected from already evaluated individuals in a previous generation. Therefore, fitness estimation is only required for individuals in newly generated offspring. However, both the parent and offspring population are passed to the subsequent selection processes to maintain the elitism in GA.

3.6.3.3 NSGA-II Selection

The core of the DSE approach in AxCGA lies in the selection process, which guides the AxCGA to identify better Pareto solutions in a design space. During an iterative DSE cycle, λ number of offspring are generated from μ number of parents in each generation. The selection process selects μ best Pareto points as new parents based on the fitness values of both the previous parents and the evolved offspring from these parents. As a result, elitism in evolution is preserved as the best-fit individuals are selected repeatedly over the DSE generations. In AxCGA, we use a fast and elitist NSGA-II, proposed by Deb et al., that selects non-dominated points in each generation and forms well-distributed Pareto-optimal solutions at the end of the DSE [225].

During each generation of the evolution, the NSGA-II performs a Pareto dominance sorting on the points based on their fitness values. The dominance relation is defined for decision vectors x and x', for which x is said to dominate x' if and only if:

$$z_i(x') \leq z_i(x)\ \forall i \in \{1, \ldots, m\} \text{ and} \tag{3.16}$$

$$z_j(x') < z_j(x)\ \exists j \in \{1, \ldots, m\}. \tag{3.17}$$

The decision vectors which are not dominated by any other vectors in the design space S are called non-dominated vectors, and with the fitness values of all the non-dominated decision vectors, a Pareto front is formed.

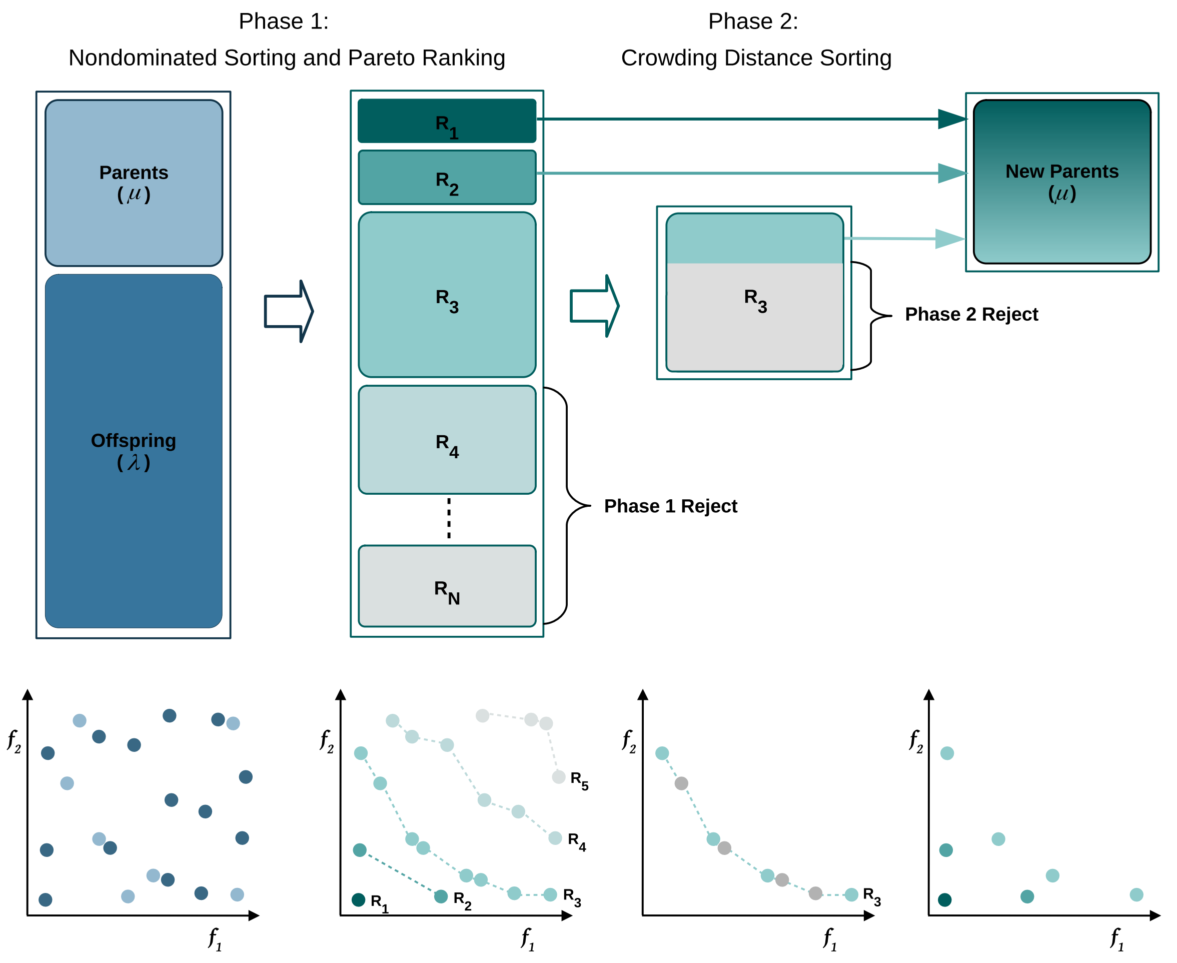


Figure 3.7: Overview of the two-phase selection process in classical NSGA-II with an example (inspired from [231])

The NSGA-II selection involves mainly two phases. Initially, all the $(\mu+\lambda)$ individuals in a combined parent and offspring population are sorted based on their dominance in fitness values. As a result, different Pareto ranks or non-domination levels from R_1 to R_N are formed where the Pareto ranks $R_1 > R_N$, and each non-domination level contains different number of individuals. During the selection process, μ number of individuals are selected for the subsequent genetic operations to form the next generation offspring, starting from the first non-dominated level R_1. However, to select precisely μ individuals from $(\mu+\lambda)$ individuals, a subset of individuals often has to be selected from an intermediate Pareto rank. We will further denote this front as *final relevant non-domination level.* Given a design space in the figure, with $\mu = 7$ and $\lambda = 14$, the first non-domination level R_1 has one point, and the second level R_2 has two points. These points from the first two levels are selected directly in Phase 1. However, to select further four points, the R_3

contains more individuals than the selection requires. Therefore, individuals from this final relevant non-domination level are passed to Phase 2 to select the required number of individuals based on crowding distance computation. All individuals in the high-ranked non-domination levels than the final relevant non-domination level are unambiguously selected for genetic operations, and all the low-ranked non-domination levels are directly rejected in Phase 1.

In Phase 2, the crowding distance is calculated for all individuals in the final relevant non-domination level to select points with maximum diversity. Figure 3.8 shows an exemplary Pareto front or non-domination level P, and the crowding distance of the i-th individual in P is estimated as

$$\Delta_i = \sum_{j=1}^{m} \frac{d_j^i}{\Delta f_j}, \tag{3.18}$$

where

$$\Delta f_j = |f_j^{max} - f_j^{min}|, j \in [1, m], \tag{3.19}$$

$$d_j^i = |f_j^{i+1} - f_j^{i-1}|, \forall i \in P. \tag{3.20}$$

The $[f_j^{max}, f_j^{min}]$ are the maximum and minimum fitness values evaluated for the j-th objective, and $[f_j^{i+1}, f_j^{i-1}]$ are the values of $i+1$ and $i-1$ points in the same objective. Since the extreme points have only one neighbor, the crowding distance of these points is assigned as ∞. During the crowding distance based selection in Phase 2, the points are selected in the order of the higher crowding distance. Therefore, the extreme points are selected first, followed by intermediate points with a higher crowding distance until the μ number of individuals are selected from both phases.

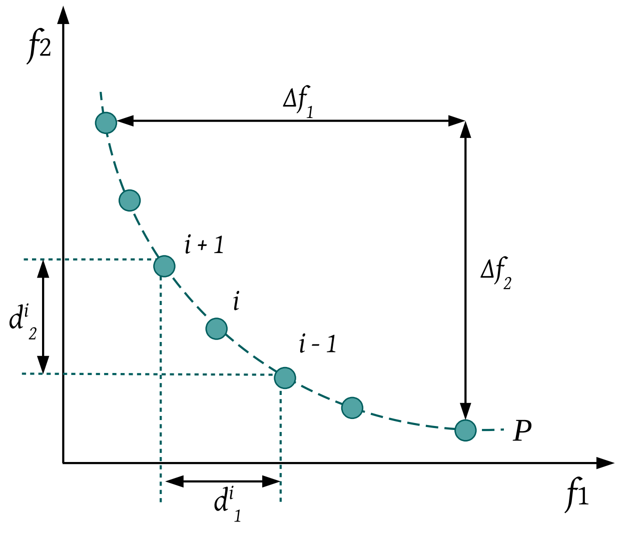


Figure 3.8: Example Pareto front for crowding distance calculation

3.6.3.4 Stop Condition

The stop condition often depends on the performance of a DSE, which can be measured from evolved Pareto-optimal configurations. The quality of a Pareto front can be accounted using a well-known metric called hypervolume indicator [230]. Hypervolume represents both convergence and diversity of Pareto solutions as a single value. During the DSE, AxCGA computes hypervolume over each generation, and often hypervolume saturates the improvements after a number of generations. Therefore, the DSE can be stopped after a number of generations without any considerable improvement in the hypervolume value. However, in each case study considered in this dissertation, we used a fixed number of generations N_{gen} as the stop condition to demonstrate the consistency of the results obtained with AxCGA. A designer can invoke a performance-based stop condition in an actual application by selecting the number of generations as `None`.

3.6.3.5 Genetic Operations

Genetic operations are performed on selected parents to produce offspring in the next generation. In AxCGA, we use genetic operations such as reproduction, mutation, and crossover. The reproduction operation randomly picks one individual from the parent population as a next-generation offspring. Mutation operation corresponds to random changes in a chromosome and results in a single offspring. In crossover operation, two random parent chromosomes mate and form two new offspring with combined genetic characteristics.

For the evolution of λ offspring from μ parents, we adopted `varOr()` methodology from the DEAP library. To perform the genetic operation, a loop iterates λ times, where each time it chooses individuals randomly from the parent population μ, and one of the genetic operations is performed. Initially, each iteration chooses a random probability value between 1 and 0. If this random probability value is less than the crossover probability p_c, a crossover operation is performed with two randomly selected individuals and keeps one of the two new individuals as a next-generation offspring. Similarly, a mutation operation is performed on a randomly selected parent individual if the random probability value is between p_c and $p_c + p_m$, where p_m is the mutation probability. In `varOr()`, $p_c + p_m$ is always defined to be less than 1. Therefore, a reproduction is performed when the random probability value is greater than $p_c + p_m$, where the reproduction probability $p_r = 1 - (p_c + p_m)$. Finally, each iteration results in one offspring, and λ number of offspring together forms an offspring population.

Among different genetic operations, the reproduction is a straightforward operation, whereas both the mutation and crossover operations need to consider the encoding style. Therefore, the mutation and crossover operations used in AxCGA are explained in detail below.

Mutation: Mutation operation in AxCGA is introduced as a generic operation in a way that it handles the parameter dependencies. Initially, with a sublist-based real-value encoding, the sublist of one of the independent pipeline stages is randomly chosen. Subsequently, one of the modules within this stage is selected if multiple modules exist. Finally, a randomly selected parameter within this module or stage is replaced with a random parameter choice by keeping the constraints of the chosen parameter. Due to the dependency propagation within a module, all the subsequent parameters affected by the mutated parameter are adjusted accordingly. As an example, a random stage 2 is selected from the equation 3.11. Among three different modules within stage 2, a random module B is selected. Finally, i^{th} parameter p_i^B is randomly chosen from the module B, where $i \in (1, ..y)$. Afterward, replace the p_i^B with one of the random feasible values, and thereafter, all the parameters from i to y are checked for dependency and adjusted accordingly.

Crossover Crossover operation is performed modularly on each application pipeline stage. Since the modules within a pipeline stage are independent, a split point can be chosen within each stage based on the number of modules, and classical crossover operations such as one-point, n-point, or uniform crossover operations can be performed. However, a designer can provide custom operations for the crossover operation considering the characteristics of a specific application stage. In addition, similar independent modules often exist on each channel in image processing applications that process multiple image channels simultaneously. In such cases, a channel-based crossover can be employed in which modules representing a channel in a parent are combined with modules corresponding to a different channel in the second parent and vice versa to form two new offspring. Relevant examples of such application-aware custom crossover can be seen in our case studies (cp. Section 3.7.1.5).

3.6.4 Adaptive GA Hyperparameters

The optimal GA hyperparameters that result in the best DSE performance are often bounded to a specific application. In AxCGA, the hyperparameter set includes both parent and offspring population size μ and λ as well as crossover and mutation probabilities p_c and p_m, and a designer can optionally specify these GA parameters. Finding the best combination of these parameters in a specific DSE problem is challenging, as the interaction between these parameters is complex. In simple applications, a hyperparameter tuning is performed to compare the effect of these parameters on the DSE and identify a promising combination. This is often achieved by performing the DSE iteratively on a set of possible parameter combinations and selecting the hyperparameter set that maximizes the DSE performance in a specific application. However, such an iterative hyperparameter tuning is not realistically feasible in complex DSE problems with large design spaces. To address this issue, many approaches use common hyperparameter values from the literature with high crossover and low mutation probabilities for better exploration and exploitation of a design space [232]. However, using such parameter values does not always give a consistent DSE performance across multiple DSE problems.

To avoid extensive hyperparameter tuning in complex DSE problems, AxCGA uses an approach that adaptively supplies the hyperparameter values during the DSE. We jointly developed this adaptive hyperparameter approach for GA, further denoted as *adaptive GA*, as part of a master thesis by Jakub Jaskólski [233]. This adaptive approach is inherited from effective mutation rate adaptation through group elite selection proposed by Kumar et al. for single-objective problems, which adapts the mutation rate based on the fitness improvement of offspring over the parents in each GA generation [234]. We modified the original approach to adapt multiple parameters with an updated feedback mechanism that handles multi-objective optimization problems. The overall idea of our adaptive GA is that hyperparameters such as crossover probability p_c and mutation probability p_m are adaptively supplied by keeping parent population size μ and offspring population size λ as constant in DSE problems.

In a typical generation, adaptive GA uses a set of different p_c and p_m and generates different number of individuals k with each parameter set. A specific p_{c_i} and p_{m_i} together with a number of required offspring k_i that have to be generated using these specific probabilities are named as metaparameters. During a DSE, together with the evolution a new offspring population, a group of metaparameters are also evolved. The metaparameters corresponds to the group which produces offspring with a best feedback score are preserved for elitism in each generation, whereas metaparameters of other groups are evolving. Combining all the individuals created from each metaparameter group together forms λ offspring in a generation.

The algorithmic flow of our adaptive GA approach is described in Algorithm 1. The adaptive initialization starts with the GA-specific initialization in AxCGA. We used commonly employed $\mu = 50$ and $\lambda = 100$ as constant parameters [232] and adapted other parameters p_c and p_m adaptively in our approach. The only input parameter required for our adaptive approach is a group size parameter σ, which determines the number of independent metaparameter groups K involved in DSE. However, this value can be specified based on μ, and does not need additional parameter tuning [234]. During the initialization, p_m is chosen as a random number linearly distributed between 0 and 1, and p_c is estimated as $1 - p_m$ for each independent group in K. As a result, the reproduction operation in `varOr()` will never occur in adaptive GA, and only mutation and crossover operations can be performed. However, due to elitist selection in NSGA-II, all individuals in the parent population are also considered for the selection process together with the generated offspring. Therefore, redundant individuals in the selection process due to the reproduction operation can be avoided in adaptive GA. Afterward, λ/K number of offspring required to be generated from each group, where the first group might contain ($\lambda \ mod \ K$) more individuals than other groups to precisely produce a total of λ individuals from all the groups. Subsequently, an initial population is also formed, and fitness is evaluated with respect to the initialized objectives.

An iterative DSE cycle starts with producing offspring O from the initial population using `varOr()` function and initialized metaparameters. The evaluated fitness values of each newly generated offspring define its offspring score S_O, and the fitness values of the parents used to produce the offspring define the parent score S_P. Since two parents are involved in the crossover operation, S_P can be estimated from one of the parents involved. Upon the generation of λ offspring, the spread of the scores are computed based on the maximum and minimum scores from a combined list of parent and offspring populations. In the subsequent step, the scores are normalized using this spread, and a feedback score S_F is computed as the difference between the parent and offspring scores after the normalization. Additionally, to calculate a single feedback score from multiple objectives, the fitness values from individual objectives are multiplied before computing the difference between the scores. A group score S_G for each specific metaparameter group is then determined as the maximum S_F among all members belonging to this group. After that, the metaparameter groups are sorted based on this group scores, and the p_c and p_m of the best group are preserved for elitism in the next generation. The probabilities p_m of remaining $K - 1$ groups for the next iteration are replaced with a randomly selected value from a triangular distribution of $[0, p_m, 1]$, and the new p_c is estimated from the new p_m similarly as before. Finally, the number of

Algorithm 1: Adaptive group elite crossover and mutation rate control mechanism

```
Input  : parent population $P$, $\mu$, $\lambda$ , group size parameter $\sigma$
Output : Pareto front $F$
1  initializations
2  $\mu = 50, \lambda = 100$, number of metaparameter groups $K = \mu^{\sigma}$ ;
3  $p_m$ =linear distribution $[0,1]$ , $p_c = 1 - p_m$ ;
4  metaparameters $M_{i \in K} = [p_{c_i}, p_{m_i}]$ ;
5  number of required offspring $k_{i=1} = (\lambda/N) + (\lambda \mod K)$ ;
6  $k_{i \in [2,..K]} = \lambda/K$ ;
7  $P = initialPopulation()$;
8  fitnessEvaluation(P);
   /* main loop representing AxCGA DSE cycles */
9  do
       /* offspring generation based on metaparameters */
10     for $i \leftarrow 1$ to $K$ do
11         for $j \leftarrow 1$ to $k_i$ do
12             offspring $O_{i,j} = varOr()$ genetic operations with $M_i$;
13             offspring score $S_{O_{i,j}} = fitnessEvaluation(O_{i,j})$;
14             parent score $S_{P_{i,j}} = getFitness$(individual in $P$ used for $varOr()$ evolution);
15         end
16     end
       /* group score estimation based on the parent and offspring scores */
17     $spread = maximum([S_P + S_O]) - minimum([S_P + S_O])$;
18     for $i \leftarrow 1$ to $K$ do
19         for $j \leftarrow 1$ to $k_i$ do
20             $S_{P_{i,j}} = (S_{P_{i,j}} - minimum([S_P + S_O]))/spread$ ;
21             $S_{O_{i,j}} = (S_{P_{i,j}} - minimum([S_P + S_O]))/spread$ ;
22             feedback score $S_{F_{i,j}} = product(S_{P_{i,j}}) - product(S_{O_{i,j}})$ ;
23         end
24         group score $S_{G_i} = maximum(S_{F_i})$;
25     end
       /* generation of new metaparameter set for non-elitist groups */
26     $sortGroups(S_G)$;
27     total score $T = sum(S_G)$;
28     for $i \leftarrow 2$ to $K$ do
29         $p_{m_i}$ = triangular distribution $([0, p_{m_i}, 1])$ ;
30         $p_{c_i} = 1 - p_{m_i}$ ;
31         $M_i = [p_{c_i}, p_{c_i}]$ ;
32         percentage contribution $X_i = (S_{G_i} - min(S_G))/T$ ;
33         $k_i = X_i * \lambda$
34     end
35     $k_{i=1} = \lambda - sum(n_{i \in [2,..N]})$ ;
36     $P = selection(P + O)$ ;
37     Pareto front $F = paretoUpdate(P)$ ;
38 while (stop condition does not meet);
39 return Pareto front $F$
```

offspring that have to be produced from each group in the next generation is estimated based on the percentage contribution of group scores. The number of additional offspring to produce exact λ offspring in a generation is added to the elite group. These new metaparameters are used for the evolution of offspring in the next cycle. The GA search continues with the selection and Pareto update operations until the stop condition is met and final configurations are evolved.

3.6.5 AxCGA Runtime Complexity

The DSE runtime often varies depending on the target applications. The overall DSE time in AxCGA can be estimated as the sum of time required for all the fitness evaluations and time to perform the algorithmic execution. Therefore, with a sequential evaluation of fitness, the time needed for the DSE t_{AxCGA} can be approximately estimated as

$$\begin{aligned} t_{AxCGA} &\approx \textit{time for fitness evaluations} + \textit{time for algorithmic execution} && (3.21)\\ &\approx [(\mu + \lambda \times N_{gen}) \times t_f] + [N_{gen} \times t_a], && (3.22)\end{aligned}$$

where μ, λ and N_{gen} are initial population size, offspring population size, and the total number of DSE generations, and t_f and t_a are the average time required for a single fitness evaluation and average algorithmic time needs to run a single DSE generation. However, due to the independent fitness evaluation of individuals in a generation, these evaluations can be parallelized based on the number threads or cores available on a computing system. Ideally, the fitness evaluations can be parallelized by a factor of $min(N_{cores}, \lambda)$, where N_{cores} is the number of available system cores. Therefore, the t_{AxCGA} can be redefined as

$$t_{AxCGA} \approx \frac{[(\mu + \lambda \times N_{gen}) \times t_f]}{min(N_{cores}, \lambda)} + [N_{gen} \times t_a]. \tag{3.23}$$

The algorithmic time t_a mainly includes the time required for the genetic operations, NSGA-II selection, and adaptive GA. Even though the fitness models used in our approach are fast enough compared to the state-of-the-art techniques, most of the time in the DSE is spent on fitness evaluations, especially for the quality evaluation using a training image set in approximate image processing applications. Therefore, t_a is the multiple orders of magnitude less than t_f, which makes $[N_{gen} \times t_a]$ negligible in real-world applications such as the case studies considered in this work. Accordingly, the t_{AxCGA} can be approximately computed as

$$t_{AxCGA} \approx \frac{(\mu + \lambda \times N_{gen}) \times t_f}{min(N_{cores}, \lambda)}. \tag{3.24}$$

3.7 Case Studies

To demonstrate our proposed AxCGA methodology, this section presents two image processing case studies that stream pixels over different processing components. These case studies are already introduced in our previous publications [23, 24, 25, 26, 27, 28, 29]. The first case study is a color space conversion, converting pixel values from one representation to another. The second case study is a display rendering application that transforms images to adopt specific characteristics of digital displays and involves a sequence of processing stages, including a color space conversion stage. These case studies demonstrate how to reuse a specific processing stage of an application in a more complex application. In digital image processing, a pipeline bitwidth often ranges from 8-bit in typical applications to 16-bit in high-end applications like motion picture cameras. Therefore, our case studies use a bitwidth $b = 12$ bits per color channel. The experiments in the following sections target implementations of these case studies on a `10AS066N3F40E2SG` FPGA from the Intel Arria 10 device family [235], with an operating frequency of 266.66 MHz and temperature of 50°C.

In the following subsections, each case study and how AxCGA is applied to these case studies are briefly explained. In addition, we describe DFG representation of applications, employed approximations and their parameter ranges, and respective design space complexities of each case study. We further explain genetic encoding and operations used to guide the AxCGA search. Finally, Pareto-optimal solutions obtained from multiple experiments are depicted, and the performance of AxCGA is evaluated and compared in different scenarios based on these results.

3.7.1 Case Study 1: *RGB* to *YCbCr* Color Space Conversion

An image color space defines and quantifies visual stimulation and is often represented with 3-different coordinates. Depending on the nature of an application, various color spaces such as RGB, LUV, $YCbCr$, HSV, and so on can be employed to represent the visual stimulation. The color spaces such as $YCbCr$ approximate visual perception uniformly where Y coordinates store luminous information independently of chromaticity information Cb and Cr, whereas RGB is more appropriate for digital devices where each channel R, G, and B represent luminance and chromaticity together. Therefore, applications sometimes require converting the colors from one representation to another. We consider such an RGB to $YCbCr$ color space conversion as the first case study to demonstrate our AxCGA approach.

An overview of RGB to $YCbCr$ color space conversion is shown in Figure 3.9. The RGB triplets in an input image are multiplied with 3-by-3 color space constants to form an output $YCbCr$ image, and Equation 3.25 formulate this conversion as matrix multiplication. Each output coordinate is independently obtained by summing up the product of RGB triplets with three different color space constants. Therefore, this application includes only arithmetic operations such as multipliers and adders.

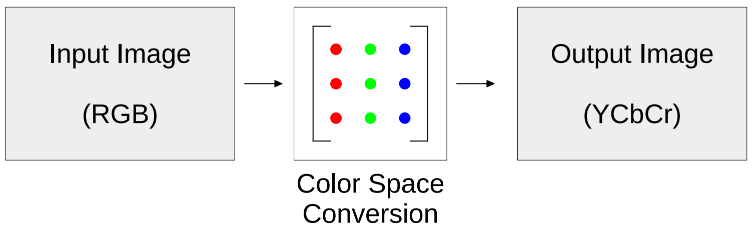


Figure 3.9: Overview of the RGB to $YCbCr$ color space conversion

$$\begin{bmatrix} R \\ G \\ B \end{bmatrix} \text{x} \underbrace{\begin{bmatrix} 0.299 & 0.587 & 0.114 \\ -0.1687 & -0.3313 & 0.5 \\ 0.5 & -0.4187 & -0.0813 \end{bmatrix}}_{\text{color space constants}} = \begin{bmatrix} Y \\ Cb \\ Cr \end{bmatrix} \tag{3.25}$$

3.7.1.1 DFG Representation

Each independent output in $YCbCr$ color space is obtained after three multiplication and two addition operations, further denoted as a channel mixer. On a hardware platform, these three multiplications can be implemented simultaneously, whereas the additions are performed in two levels in each channel mixer. The RGB to $YCbCr$ conversion mentioned in Equation 3.25 comprises three independent channel mixers, and hardware implementation resulted in total 9 different multiplications and 6 addition operations. During AxCGA initialization phase, we create DFG for a single channel mixer which can be further replicated for other channels since mixer components are independent of input values. The resulting application DFG is shown in Figure 3.10, and it contains scalable arithmetic operations implemented in fixed-point arithmetic.

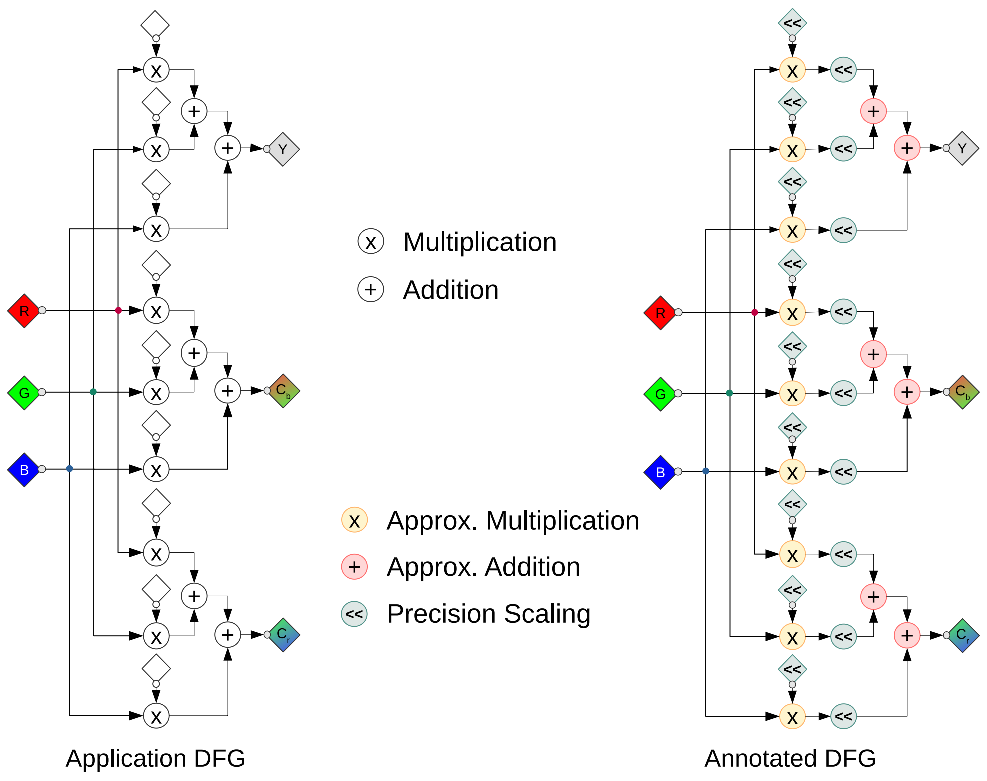


Figure 3.10: Application DFG and annotated DFG of the RGB to $YCbCr$ color space conversion

3.7.1.2 Approximation Techniques and Parameters

To create RGB to $YCbCr$ conversion designs with reduced power consumption, we apply suitable approximation methods for each intermediate node $n \in N$ of the application DFG. Approximation choices are incorporated during annotation of the application DFG, and the annotated DFG with specific approximation nodes is also shown in Figure 3.10. The nodes corresponding to arithmetic operations, such as adders and multipliers, are replaced with their approximate variants. Additionally, precision of intermediate signals resulting from each multiplication and the fractional part of each matrix coefficient are scaled to approximate the design further.

Multiple variants of approximate arithmetic components are selected from the approximate component library described in Section 3.4. An overview of different approximate computing techniques that are employed to RGB to $YCbCr$ color space conversion is listed in Table 3.2. These methods expose different parameters that have to be optimized in conjunction to determine the quality-power trade-off effectively. The following sections briefly describe each approximate computing technique employed in this case study and their possible approximation parameters.

Table 3.2: List of approximation parameters of RGB to $YCbCr$ color space conversion

Parameters	**Symbols and Ranges**	**Instances** Modular Level (i) Node Level (j)
Fractional bits of matrix coefficients	$F_{\text{co}}(i,j) \in [0, F_{\text{co,ref}}]$, where $F_{\text{co,ref}} = 13$	$i = \{1,2,3\}$ $j = \{1,2,3\}$
multiplier types	$M_{\text{t}}(i,j) \in \{\texttt{Acc}, \texttt{BAM}\}$	$i = \{1,2,3\}$ $j = \{1,2,3\}$
BAM HBL	$M_{\text{h}}(i,j) \in \begin{cases} \left[0, \frac{\text{maxHBL}(i,j)}{2}\right], & \text{if } M_{\text{t}}(i,j) = \texttt{BAM} \\ \{\text{ignored}\}, & \text{otherwise} \end{cases}$	$i = \{1,2,3\}$ $j = \{1,2,3\}$
BAM VBL	$M_{\text{v}}(i,j) \in \begin{cases} \left[0, \frac{\text{maxVBL}(i,j)}{2}\right], & \text{if } M_{\text{t}}(i,j) = \texttt{BAM} \\ \{\text{ignored}\}, & \text{otherwise} \end{cases}$	$i = \{1,2,3\}$ $j = \{1,2,3\}$
Fractional bits of intermediate results	$F_{\text{in}}(i,j) \in [0, \max(F_{\text{co}}(i,[1,3]))]$	$i = \{1,2,3\}$ $j = \{1\}$
adder types	$A_{\text{t}}(i,j) \in \{\texttt{Acc}, \texttt{LSA}, \texttt{MA}\}$	$i = \{1,2,3\}$ $j = \{1,2\}$
adder split points	$A_{\text{p}}(i,j) \in \begin{cases} [0, (b + F_{\text{in}}(i))], & \text{if } A_{\text{p}}(i,j) \in \{\texttt{MA}, \texttt{LSA}\} \\ \{\text{ignored}\}, & \text{otherwise} \end{cases}$	$i = \{1,2,3\}$ $j = \{1,2\}$
LSA input select	$A_{\text{s}}(i,j) \in \begin{cases} [0,1], & \text{if } A_{\text{t}}(i,j) = \texttt{LSA} \\ \{\text{ignored}\}, & \text{otherwise} \end{cases}$	$i = \{1,2,3\}$ $j = \{1,2\}$

Precision Scaling

Precision scaling is a simple approximation technique in which the precision of data or signal varies by changing its bitwidth, as explained in Section 2.1.3.1. The matrix coefficients have fractional parts in the RGB to $YCbCr$ conversion case study, and multiplying such a fractional number with an integer pixel value is also resulted in fractional parts. Therefore, we vary the fractional bitwidth of each matrix coefficient F_{co} independently between 0 and reference precision $F_{co,ref}$. Similarly, we vary the fractional bitwidth of intermediate signals from the multiplication operation between 0 and the maximum precision F_{co} within each channel mixer. The reduction in precision leads to a smaller number of bits used for the signals, which reduces the number of registers in FPGA designs and often decreases the size of subsequent arithmetic operations.

Approximate Multipliers

To perform AxCGA experiments, we consider both an approximate multiplier component and an accurate multiplier *Acc*. The accurate multiplier can be either implemented using DSPs or with LUT logic, depending on the lower power consumption for the incoming input bitwidths and target frequency. Based on the characterization and comparison of different approximate multipliers, the BAM outperformed all other alternatives in the approximate component library. Therefore, we consider only BAM as an approximate multiplier in this case study.

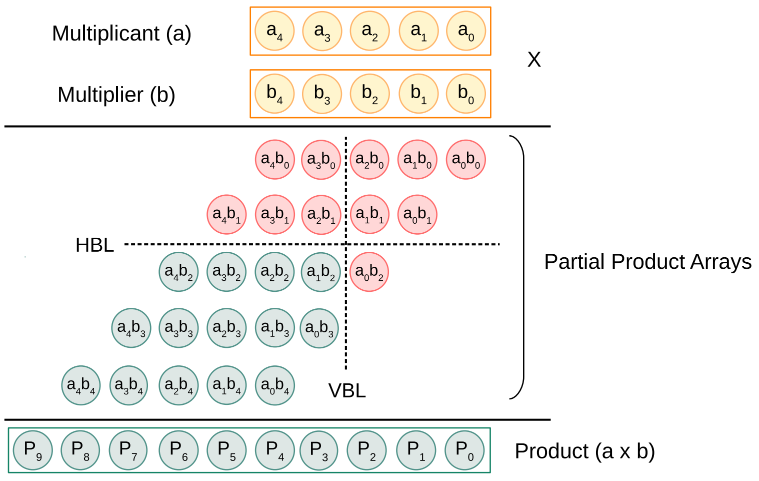
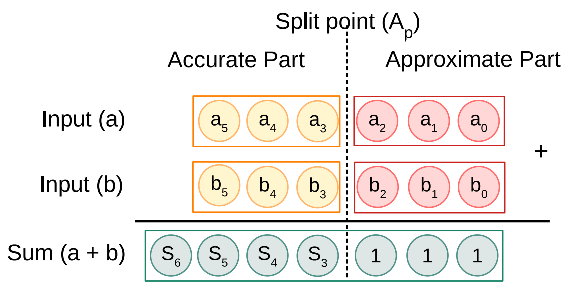

Multiplicant (a)
Multiplier (b)
X
HBL
VBL
Partial Product Arrays
Product (a x b)
a_4 a_3 a_2 a_1 a_0
b_4 b_3 b_2 b_1 b_0
a_4b_0 a_3b_0 a_2b_0 a_1b_0 a_0b_0
a_4b_1 a_3b_1 a_2b_1 a_1b_1 a_0b_1
a_4b_2 a_3b_2 a_2b_2 a_1b_2 a_0b_2
a_4b_3 a_3b_3 a_2b_3 a_1b_3 a_0b_3
a_4b_4 a_3b_4 a_2b_4 a_1b_4 a_0b_4
P_9 P_8 P_7 P_6 P_5 P_4 P_3 P_2 P_1 P_0

Figure 3.11: Overview of BAM multiplier

BAM is an approximate PP accumulation technique used to approximate a multiplier as described in Section 2.1.3.3. In BAM, the PP arrays as a result of a binary level multiplication, are truncated both horizontally and vertically using HBL and VBL. Therefore, the approximation parameters are HBL and VBL, where the HBL configures the approximation in coarse grain, and VBL is used for finer approximations. The dot diagrams in Figure 3.11 illustrate the concept of BAM multiplier on the binary level. The input sizes of the multiplier determine the ranges of approximation parameters. The range of HBL depends on the number of PP arrays, which is determined by the bitwidth of the smaller input. Similarly, the VBL range depends on the size of the product array, which is determined by the sum of input bitwidths. As an additional constraint, the VBL should be greater than or equal to HBL to exclude irrelevant configurations. In our case study, we restrict the ranges of HBL and VBL to half of the maximum possible value to avoid high errors and thereby restrict the size of the design space meaningfully.

Approximate Adders

We use two approximate adders such as MA and LSA together with an accurate adder as multiple adder choices in our case study. These approximate adders are multiple-bit approximate adder type, where the approximation techniques consider a number of bits or a segment of adders together (cp. Section 2.1.3.2). These are selected based on the characterization and comparison of various approximate adders presented in [216]. Such a multiple-bit approximate adder exposes a parameter A_p to split an addition into an approximate and an accurate part, as shown in Figure 3.12. In an MA, the output from the approximate part is replaced with a median value if the input data distribution is known. Otherwise, all the output sum bits are directly set to binary 1. In addition, the carry bit from the approximate part is set as zero. In an LSA adder, an additional LSB input select parameter A_s directly assigns one of the inputs as the sum of the approximate part. The MSB of the non-selected approximate part is picked as the carry-bit from the approximate to the accurate part. The ranges of split points are determined by the pipeline bitwidth b and intermediate fractional bitwidth F_{int}.


Split point (A_p)
Accurate Part
Approximate Part
Input (a)
Input (b)
Sum (a + b)
a_5 a_4 a_3 a_2 a_1 a_0
b_5 b_4 b_3 b_2 b_1 b_0
+
S_6 S_5 S_4 S_3 1 1 1
(a) Median adder (MA)

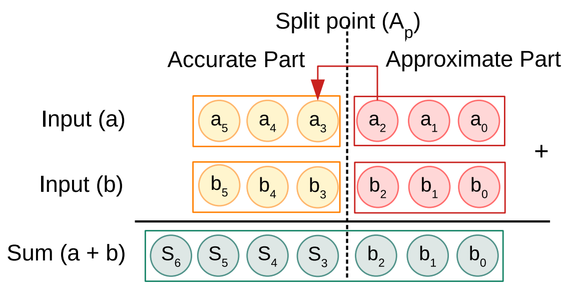

Split point (A_p)
Accurate Part
Approximate Part
Input (a)
Input (b)
Sum (a + b)
a_5 a_4 a_3 a_2 a_1 a_0
b_5 b_4 b_3 b_2 b_1 b_0
+
S_6 S_5 S_4 S_3 b_2 b_1 b_0
(b) LSA adder

Figure 3.12: Approximate adders used in the RGB to $YCbCr$ color space conversion case study

3.7.1.3 Design Space complexity

Design space complexity λ of an approximate computing application can be defined as the total number of feasible approximation parameter combinations in an application. Therefore, this defines the possible number of candidate DFGs an annotated DFG produces. Considering the approximation parameters and their ranges in Table 3.2, the design space complexity of a 12-bit channel mixer is

$$\lambda_{\text{Channel Mixer}} \approx (4.225 \times 10^{12}) \text{ or } (4.808 \times 10^{12}). \tag{3.26}$$

These numbers vary slightly due to the difference in the signedness of matrix coefficients in each mixer. Therefore, the overall design space complexity of a three-channel RGB to $YCbCr$ conversion is

$$\lambda_{RGB \text{ to } YCbCr} \approx 9.766 \times 10^{37}. \tag{3.27}$$

3.7.1.4 Genetic Encoding

RGB to $YCbCr$ conversion is a single-stage application with three independent channel mixer modules, as shown in Figure 3.10. Therefore, the genetic encoding initially groups these three module instances at the stage level with real-value genetic encoding, i.e., the parameters from the three channel mixers.

$$\text{Individual } I_{(\text{RGB to YCbCr})} = [P^{(\text{RGB to YCbCr})}] \tag{3.28}$$

$$= [P^1_{mixer}, P^2_{mixer}, P^3_{mixer}] \tag{3.29}$$

The second-level sub-list groups all the interdependent node types within each channel mixer module by considering the priority order of parameter propagation. After that, a third-level list combines the independent instance of similar nodes with a single module. Considering approximation techniques specified in Table 3.2, the overall parameters of a channel mixer module can be encoded as

$$P_{mixer} = [[F^1_{\text{co}}, F^2_{\text{co}}, F^3_{\text{co}}], [M^1, M^2, M^3], F_{\text{in}}, [A^1, A^2]]. \tag{3.30}$$

where an approximate multiplier M and approximate adder A exposes additional parameters and can be encoded as

$$M = [M_t, [M_h, M_v]], \tag{3.31}$$

$$A = [A_t, [A_p, A_s]]. \tag{3.32}$$

For an easier representation of these parameters in our AxCGA implementation, we encode approximate multipliers $M_t(i, j)$ as `Acc` = 0 and `BAM` = 2. Similarly, we encode approximate adder type $A_t(i, j)$ as `Acc` = 0, `LSA` = 2, and `MA` = 4. The remaining numbers from 0 to 6 are used to encode other available multipliers and adders in the library, which are not used in this case study.

An example real-value encoded individual based on the approximation parameters and their ranges listed in Table 3.2 can be represented as

$$\begin{aligned} I_{(\text{RGB to YCbCr})} = \quad & [[[5, 6, 5], [[2, 2, 7], [0], [2, 0, 5]], 5, [[0], [2, 16, 0]]], && \rightarrow P^1_{mixer} \\ & [[8, 6, 2], [[0], [0], [2, 2, 8]], 0, [[4, 6], [2, 9, 0]]], && \rightarrow P^2_{mixer} \\ & [[9, 2, 5], [[2, 1, 11], [2, 1, 8], [0]], 0, [[2, 11, 1], [4, 8]]]] && \rightarrow P^3_{mixer} \end{aligned}$$

3.7.1.5 Genetic Operations

Genetic operations such as mutation and crossover are performed on randomly selected individuals to create offspring based on specific mutation probability p_m and crossover probability p_c.

A mutation operation selects a random node from all the existing nodes in three mixer modules. Then, the parameters of this node are replaced with different parameters randomly chosen from their possible ranges. Finally, subsequent dependent nodes or parameters are also adjusted accordingly. For example, if M_t^1 is chosen for mutation operation, it is replaced by either Acc or a BAM multiplier with new M_h^1, and M_v^1. Due to the priority order of parameter dependencies,

$$Priority\ F_{co}^1 > Priority\ M^1 > Priority\ F_{in}^1 > Priority\ A^1, \tag{3.33}$$

the new M_h^1, and M_v^1 are chosen depending on the F_{co}^1, and the following parameters F_{in}^1 and A^1 within the first channel mixer are also adjusted accordingly.

A crossover operation combine two parent individuals, and we used a single-point crossover operation to generate offspring. Since RGB to $YCbCr$ conversion is a single stage 3 channel image processing application, it contains three similar modules corresponding to each channel mixer. Therefore, we use a channel-based crossover operation, which applies a single-point crossover on a modular basis that splits channels from parents and recombines different channels to generate offspring.

For two individuals I_1 and I_2 with three channel mixers in Figure 3.13, an example channel-based crossover produces two offspring O_1 and O_2, as shown in the figure.

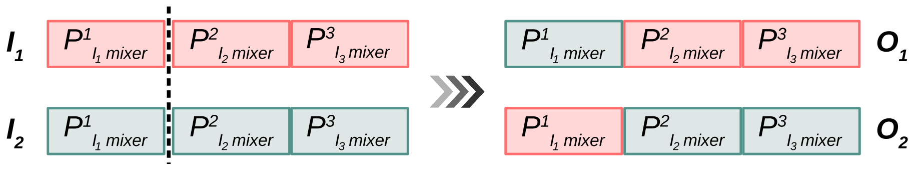


Figure 3.13: Channel-based single-point crossover

3.7.1.6 DSE Objectives

The DSE on this case study aims to trade off the application quality for better system power consumption. As described in Section 3.5.1.2, the total power consumption of an individual corresponding to a typical parameter configuration is estimated from the total number of FPGA resource units. Since the power consumption in this application indirectly reflects the area of the design, we use power as an objective among these two values in optimization. However, AxCGA keeps track of the area consumption from the models to ensure that the estimated area is always within the available FPGA resources on a targeted device.

$YCbCr$ is a standard color space for digital videos and often used for MPEG compression in DVDs, digital TVs, and Video CDs. With approximate computing, the quality degradation in $YCbCr$ can be estimated based on the loss of information between images obtained with a reference and approximate implementation. Therefore, we consider widely used PSNR, estimated based on the ratio between reference and approximate images, for the quality estimation.

The DSE objectives in approximate computing are always maximizing the application quality while minimizing the power consumption. Therefore, the objective function is defined as

$$f_{obj} = minimize(\text{power}) \wedge maximize(\text{PSNR})\,. \tag{3.34}$$

3.7.1.7 AxCGA Initialization and Setup

A DSE experiment starts with both the application-specific and GA-specific initialization in AxCGA. In the application-specific initialization phase, we created an application DFG of RGB to $YCbCr$ conversion and annotated it with the approximation techniques mentioned in the previous section. We used a reference implementation of the system without any approximations to supply parameters such as toggle rate and per-unit power consumption for the power model. Simulation of the reference implementation using different images in the ARRI image set resulted in a toggle rate ranging from 0.25 to 0.33. Therefore, we selected 0.3 as an average toggle rate for the power model. In addition, we used average synthesis results from four random approximate system configurations and the reference system for the per-unit power estimation. The quality metric PSNR and relevant training dataset are initialized to estimate quality degradation during the DSE.

The selection of training data depends on the design objectives and characteristics of a specific application. For a DSE performed for an average quality value from all the processed pixels, the input training set should cover the images with specific pixel and noise distribution similar to real-world images. This could embed the influence of these distributions into the decision-making from the DSE results. In contrast, if the DSE is performed for the worst-case quality from a single pixel, consideration of such distributions in the training data is not necessary, whereas the training data should possibly cover the entire input space. Since the RGB to $YCbCr$ conversion uses PSNR, which relates to an average quality from different pixels, we used real-world images for the quality estimation during the DSE.

This dissertation uses the ARRI image set captured with an ARRI ALEXA camera, which includes 12 different high-quality real-world color image sequences [238]. For the quality estimation during the DSE, we used *Color Wheel* image from the ARRI image set shown in Figure 3.14. Since these images are available in a high resolution of 2880 × 1620, using such a high-resolution image for multiple quality evaluations makes the DSE prohibitively long. Therefore, we randomly sampled 64^3 pixels from the Color Wheel image to generate training dataset for the DSE. However, all pixels from the remaining images are used to validate selected configurations obtained from the DSE.

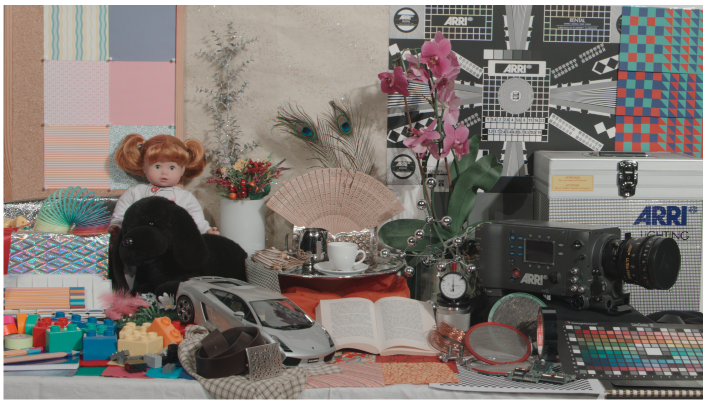


Figure 3.14: Color Wheel image from the ARRI image dataset [238]

For the GA-specific initialization, adaptive version of AxCGA is selected, and therefore, μ and λ are chosen as 50 and 100 by default. The group size parameter σ is chosen as 5/8, which resulted in 11 metaparameter groups. The parameters are encoded by considering the design constraints and node dependencies, and an initial population is formed with one individual equivalent to the reference design and all other individuals with randomly chosen parameters. The total number of individuals in the initial population also equals μ. We terminate the DSE experiments at $N_{gen} = 750$, and this value has been selected based on an empirical analysis of convergence from individual experiments. Therefore, a DSE experiment resulted in a maximum of 75 050 fitness evaluations, including the initial population, and helps to analyze and compare the performance of multiple experiments on the same scale.

3.7.1.8 DSE Results

This section explains the AxCGA results on the RGB to $YCbCr$ color space conversion application. Initially, Pareto-optimal solutions obtained from the DSE are discussed. Thereafter, a set of configurations is selected from a single DSE run and is validated for the loss of quality on a test dataset. In addition, resource consumption such as area, power, and speed are estimated for each configuration after synthesis and place & route for the targeted device.

Pareto-optimal Solutions

Figure 3.15 shows Pareto-optimal solutions obtained from 50 independent AxCGA experiments. These solutions offer a wide range of choices to a designer for selecting the desired application quality and power trade-off in the application. The number of solutions obtained from each independent experiment varies between 205 and 291, with an average of 246.76 solutions per experiment.

The maximum quality of a reference RGB to $YCbCr$ design without any error is ∞ in PSNR. However, we approximated this maximum quality value to 137.23 dB PSNR with a theoretical pixel difference of 0.5 in one channel. In all experiments, at least one configuration with maximum PSNR is present due to a reference design added in the initial population. However, it is often difficult to find many configurations in a region close to the quality of the reference design, as can be seen in the figure, due to logarithmic computation in PSNR.

Since GA is a non-deterministic optimization approach, each experiment might produce slightly different configurations. However, the overall trend of Pareto solutions is identical across each experiment. A randomly chosen experimental result with 258 solutions is highlighted in the plot to demonstrate the trend obtained with a single DSE run, and it can be seen that the obtained solutions from a single experiment are also widely spread across the solution space. Therefore, a designer is able to make a good design decision from a single DSE run. However, a few different runs are recommended for practical implementation to increase the density of the design choices all over the solution space.

DSE Results Validation

Since we use a subset image as training data for quality estimation and simple models for the resource estimation in AxCGA, the solutions obtained from the DSE need to be validated to demonstrate the suitability of our AxCGA methodology in a real-world application. A quality of 30 dB PSNR is considered as a lower acceptable quality level in many state-of-the-art approximate

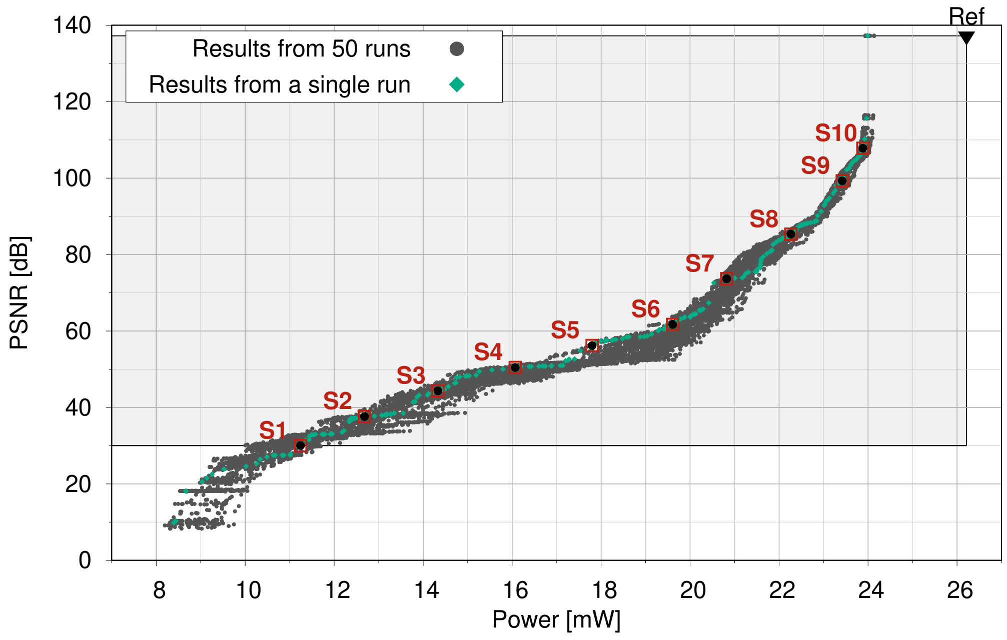


Figure 3.15: Pareto fronts obtained from the 50 independent DSE experiments on *RGB* to *YCbCr* conversion, and one randomly selected DSE front is highlighted. The inset shows the region of interest, and the reference design without any approximation is indicated as Ref.

image processing applications [14]. Therefore, to validate our AxCGA approach, we selected 10 representative solutions from a region above this quality-threshold, where a designer can realistically select the configurations in the *RGB* to *YCbCr* conversion, as shown in Figure 3.15. The selected solutions are tested for quality using all the images in the ARRI dataset, excluding the image used in training. Similarly, we synthesized these solutions and estimated resources such as power, area, and speed after place & route for the target FPGA using Intel Quartus Prime Software [236] which includes Power Analyzer [220] and Timing Analyzer [237]. These resources are further denoted as post-synthesis resources in the following sections.

Table 3.3 list validated quality results and post-synthesis resources together with the model estimated values during the DSE. From the power consumption results, even though the average error is 16.18%, the DSE estimated results are comparable to the post-synthesis results with 100% fidelity. With the selected solutions, the post-synthesis power saves between 24.15% to 59.30% in comparison to the reference design with 25.80 mW power. We additionally list the FPGA resource utilization from the synthesis results to show how many resources are saved compared to the reference design. Even though the speed of the approximate design is not an objective in the DSE, we evaluated the speed of the selected configurations also to check whether the corresponding designs satisfy the target frequency of 266.66 MHz. The estimated speed in the table shows that all the selected configurations can operate at a higher frequency than the desired target. Finally, to implement reliable approximate designs, it is important to validate the loss of application quality due to approximations on a test dataset. The quality validation results show that the PSNR values

Table 3.3: Resource usage, power consumption and quality data of the selected points

Selected	**Power**		**Quality**		**FPGA Resources**			**Speed**
Points	AxCGA	Synth.	Train	Test	DSP	LUT	Reg	Synth.
[Index]	[mW]	[mW]	[dB]	[dB]	[Units]	[Units]	[Units]	[MHz]
S1	11.24	10.50	30.01	26.87	0	180	476	334.22
S2	12.68	10.79	37.61	38.57	0	196	530	400.32
S3	14.33	12.35	44.32	43.31	0	248	569	300.12
S4	16.07	15.06	50.44	46.75	1	217	594	291.46
S5	17.80	15.74	56.19	54.68	0	342	654	299.58
S6	19.61	16.62	61.70	58.53	3	232	636	294.81
S7	20.82	17.39	73.67	72.01	3	252	670	292.40
S8	22.27	18.57	85.34	84.93	4	229	725	270.71
S9	23.42	19.24	99.27	98.11	4	253	782	271.52
S10	23.89	19.57	107.82	106.05	4	259	807	271.96
Ref	26.21	25.80	–	–	9	275	842	272.16

on the test data satisfy the acceptable quality-threshold from S2 to S10 configurations. From the table, it can be seen that the percentage error in PSNR varies between 11.68% and 0.48% with an average of 3.82%. In all the selected configurations except S2, the average test quality is slightly less than the training quality. Therefore, selecting a configuration from the DSE results with slightly higher quality than the desired quality level is recommended for practical implementations.

3.7.2 Case Study 2: Display Rendering Pipeline

We considered a display rendering pipeline as the second case study to demonstrate our AxCGA approach. The display rendering pipeline adapts the image pixels encoded as close to the actual scene while capturing to display on monitors suitably. This case study is a pixel streaming application where individual pixels are processed independently without any spatial-temporal correlation between the adjacent pixels or frames and often used in digital cameras to adjust the colors to the specification of the target monitor in terms of dynamic range, color space, and electro-optical transfer function (EOTF). The display rendering pipeline includes three different processing stages, as shown in Figure 3.16.

The first tone mapping and third EOTF compensation stages are nonlinear transformations of pixel values. Due to the higher computational complexity of nonlinear functions, these functions are implemented using lookup tables on an FPGA and can be considered for algorithm-level table-based approximations. The second color space conversion stage has a similar structure to our previous case study. In fact, this stage converts the pixels into a different color space in the display rendering application. Therefore, arithmetic components such as approximate adders and multipliers and application-level precision scaling can be used for approximating this stage.

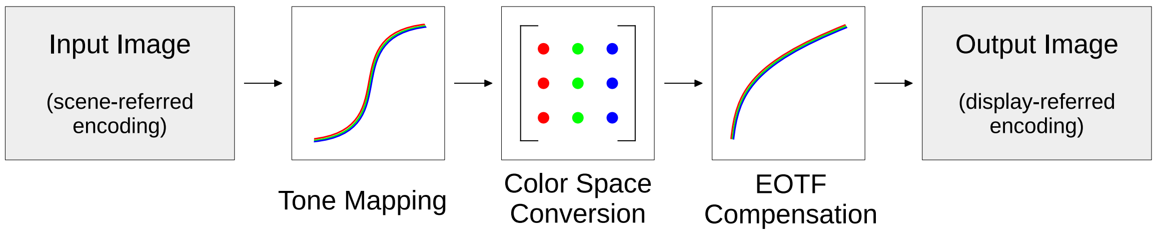


Figure 3.16: Overview of display rendering pipeline

Overall, the application of AxCGA methodology in this case study demonstrates how to integrate approximate computing techniques on multi-level abstractions and how to reuse an application DFG and genetic operations in complex applications. The following sections describe each pipeline stage in detail and different approximation techniques employed on multiple abstraction layers.

Tone Mapping

Tone mapping is the first stage in a display rendering pipeline, which transforms the input luminance of a scene-referred encoded image for a natural reproduction of scenes on higher dynamic range displays. In this case study, we use a global sigmoidal operator as proposed by Reinhard and Devlin [239], and the tone-mapped values can be computed as

$$X_{out} = \frac{\mathrm{enc}^{-1}(X_{in})}{\mathrm{enc}^{-1}(X_{in}) + (hI_a)^k} u + v, \tag{3.35}$$

where $X \in \{R, G, B\}$ represents the input or output luminance of a color channel, and h, k, and I_a represent the model parameters to control the properties such as luminance and contrast. Additionally, the constants u and v are used to scale and shift the values to a desired dynamic range. Since incoming pixels are non-linearly encoded and the tone mapping function is defined for linear intensities, this phase initially performs decoding of pixel values using an inverse encoding operation $\mathrm{enc}^{-1}()$. We use logarithmically encoded scene-referred images as inputs to the display rendering application. Therefore, an inverse logarithmic transformation $\mathrm{enc}^{-1}() = 2^{X_{log}}$ is used as decoding function. For other tone mapping function parameters, we chose $h = 9, k = 0.6, I_a = 0.4, u = 4137$ and $v = -7.4797$ in this case study. The corresponding nonlinear function can be seen in Figure 3.18a.

Color Space Conversion

The second stage of the display rendering pipeline is a 3-by-3 matrix multiplication that converts the image pixels from Alexa Wide Gamut (AWG) to sRGB color space [238]. The underlying FPGA implementation of this stage is similar to RGB to $YCbCr$ conversion case study, and only the color space constants are different. The AWG to $sRGB$ conversion is represented as

$$\begin{bmatrix} R_{in} \\ G_{in} \\ B_{in} \end{bmatrix} \text{x} \underbrace{\begin{bmatrix} 1.4850 & -0.4012 & -0.0838 \\ -0.0337 & 1.2829 & -0.2492 \\ -0.0108 & -0.1220 & 1.1112 \end{bmatrix}}_{\text{color space constants}} = \begin{bmatrix} R_{out} \\ G_{out} \\ B_{out} \end{bmatrix} \tag{3.36}$$

EOTF Compensation

The last stage of the pipeline is an EOTF compensation of the target display. The EOTF is a nonlinear function that transforms digital luminance signals into desired optical signals to display on specific monitors. However, incoming luminance signals are required to be equalized for specific target display before the EOTF conversion. In the EOTF compensation phase, this can be achieved by applying an inverse transformation of the EOTF. In this case study, we use the transfer function defined for sRGB in IEC 61966-2-1 Amendment 1 [240]:

$$X_{out} = \begin{cases} 12.92\, X_{in} & \text{, for } X_{in} < X_{th}, \\ 1.055\, X_{in}^{1/2.4} - 0.055 & \text{, for } X_{in} \geq X_{th}, \end{cases} \tag{3.37}$$

where $X \in \{R, G, B\}$ denotes the luminance of each channel and $X_{th} = 0.0031308$. Figure 3.18b shows the consequent EOTF compensation function.

3.7.2.1 DFG Representation

Each stage in an application pipeline is independently treated in AxCGA to ensure the modularity and reusability of a stage in other applications. Since the implementation of the AWG to $sRGB$ color space conversion is similar to the RGB to $YCbCr$ case study, the DFG of this stage is identical in Figure 3.10. Additionally, six independent look-up tables, each representing a nonlinear transformation for a specific image channel, are used to implement tone mapping and EOTF compensation. Therefore, each of these transfer functions serves an independent node in the DFG. The application DFG of the display rendering pipeline can be seen in Figure 3.17.

3.7.2.2 Approximation Techniques and Parameters

In this application, each intermediate node $n \in N$ is considered for suitable approximation techniques. Since the underlying operations of AWG to $sRGB$ conversion are similar to the RGB to $YCbCr$, we considered the same approximate arithmetic components and their parameter ranges listed in Table 3.2. Therefore, the choices of adder components are MA or LSA and multipliers are BAM in addition to the accurate adder and multiplier Acc. Additionally, precision scaling on an application level is applied at different places, as shown in Figure 3.17. The tone mapping and EOTF compensation are implemented with lookup tables in a reference FPGA implementation. Therefore, each lookup table contains 2^{12} table entries for all input values in a 12-bit pipeline, demanding a higher number of BRAMs in the reference implementation. To reduce this BRAM utilization, we considered a two-level hierarchical segmentation proposed by Lee et al. [67], which

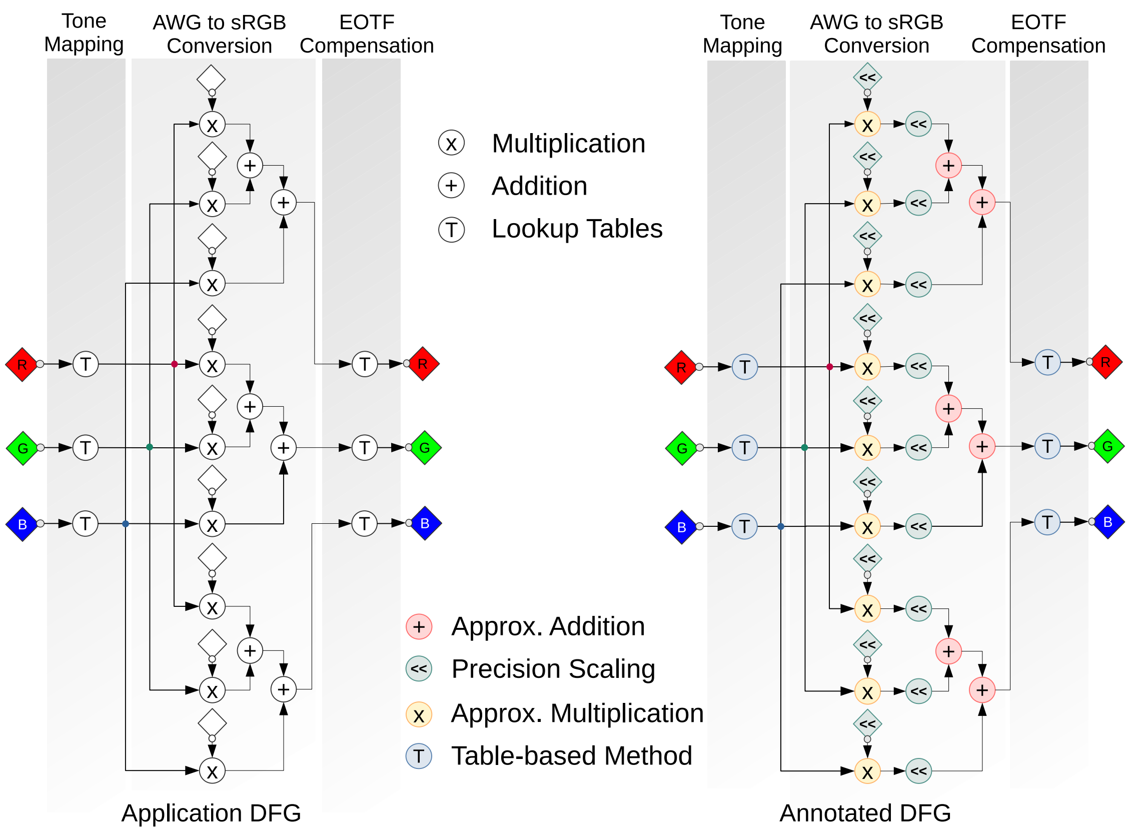


Figure 3.17: Overview of a display rendering application

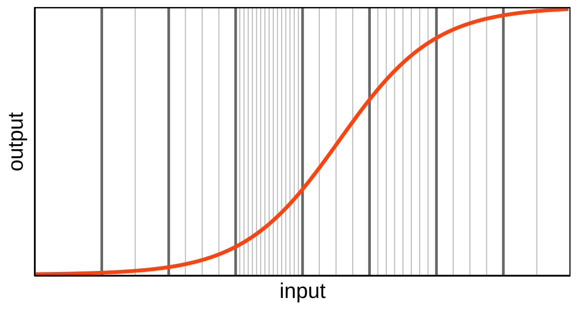
stores only a few entries correlating to the slope of an underlying transfer function. The following section describes this hierarchical segmentation in detail.

Hierarchical segmentation

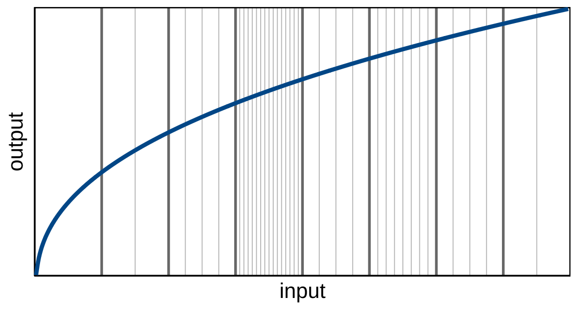
Hierarchical segmentation is used to approximate nonlinear functions, which are implemented using a lookup table. This approach saves memory usage by storing a reduced number of entries and can be synonymously named as a sparse lookup table. The display rendering case study uses a two-level hierarchical segmentation approach for implementing nonlinear functions. In the two-level segmentation, initially, the input range is split into uniformly distributed sections N_{sec}. Thereafter, each section $i \in [1, N_{sec}]$ is further divided into uniform sub-segments $N_{seg}(i)$ in the second level segmentation. Therefore, the upper limit for each sub-segments is defined by the N_{sec}. To limit the lookup table address mapping overhead in an FPGA implementation, both the N_{sec} and N_{seg} values are constrained to a power of two. With the sparse lookup table, only the values corresponding to each sub-segments are required to be stored, irrespective of values for all the inputs. Therefore, the total number of values in a sparse lookup table can be estimated directly from the approximation parameters as

$$N_{\text{total}} = \sum_{i=1}^{N_{\text{sec}}} N_{\text{seg}}(i). \tag{3.38}$$

In the display rendering case study, we used sparse lookup table to approximate both tone mapping and EOTF compensation functions, and Figure 3.18 shows an exemplary segmentation of these transfer functions. In two-level segmentation, $N_{sec} = 8$, shown as thick lines, and each section $i \in [1, 8]$ is further equally divided into $N_{seg} = [1, 2, 4, 16, 4, 8, 4, 2]$.



(a) Transfer function used in tone mapping



(b) Transfer function used in EOTF compensation

Figure 3.18: Exemplary hierarchical segmentation ($N_{sec} = 8$ and $N_{seg} = [1, 2, 4, 16, 4, 8, 4, 2]$) on the nonlinear functions used in display rendering case study

In our AxCGA experiments, we limited the upper value of N_{sec} as 32 based on our preliminary investigations to avoid non-useful parameter combinations. Additionally, we restrict the lower bound of N_{total} to 16 to avoid any configurations with higher quality loss. Besides, a linear interpolation technique is also optionally used to reconstruct the desired values more accurately. Table 3.4 lists the approximation parameters and their ranges from a sparse lookup table.

Table 3.4: List of approximation parameters of a sparse lookup table

Parameters	**Symbols and Ranges**	**Instances** Modular Level (i) Node Level (j)
Interpolation types	$T_{int}(i) \in \{None, Linear\}$	$i = \{1\}$
No. of sections	$N_{\text{sec}}(i) = 2^p$, where $p \in [0, 5]$	$i = \{1\}$
No. of sub-segments	$N_{\text{seg}}(i, j) = 2^{q_j}$, where $q_j \in [0, \log_2(\text{sizeof}(N_{sec}))]$, $\sum_{j=1}^{N_{\text{sec}}} N_{\text{seg}}(j) \geq 16$	$i = \{1\}$ $j = \{1, \ldots, N_{sec}(i)\}$

3.7.2.3 Design Space Complexity

Combing multiple pipeline stages in an application exponentially increases the number of feasible parameter configurations. Therefore, the total design space complexity λ of a display rendering application can be estimated as

$$\lambda_{\text{Display Rendering}} = \lambda_{\text{Tone Mapping}} \times \lambda_{\text{Color Space Conversion}} \times \lambda_{\text{EOTF Compensation}}. \tag{3.39}$$

Since the tone mapping and EOTF compensation stages use similar segmentation schemes, the design space complexity is identical in both cases. In addition, we use a single instance of a sparse lookup table in three channels for each tone mapping and EOTF compensation in our AxCGA experiments. For a pipeline bitwidth $b = 12$, the design space complexity of the sparse lookup table from a tone mapping or EOTF compensation stage can be estimated as

$$\lambda_{\text{Sparse Lookup Table}} \approx 1.585 \times 10^{29}. \tag{3.40}$$

Therefore, based on equation 3.40 and 3.26, the design space complexity of the display rendering case study can be computed as

$$\lambda_{\text{Display Rendering}} \approx (1.585 \times 10^{29}) \times (1.111 \times 10^{38}) \times (1.585 \times 10^{29}), \tag{3.41}$$

$$\approx 2.791 \times 10^{96}. \tag{3.42}$$

3.7.2.4 Genetic Encoding

The display rendering application has three stages in the processing pipeline, and a typical individual is encoded as

$$\text{Individual } I_{\text{(Display Rendering)}} = [P^{\text{(Tone Mapping)}}, P^{\text{(Color Space Conversion)}}, P^{\text{(EOTF Compensation)}}]. \tag{3.43}$$

The genetic encoding of $P^{\text{(Color Space Conversion)}}$ is identical to the $P^{\text{(RGB to YCbCr)}}$ as in Equation 3.29. Since both tone mapping and EOTF compensation stages use a unique instance of sparse lookup tables for approximations, only two instances of the sparse lookup table occur at an application level. However, these are distinct stages in the processing pipeline, and corresponding instances are, therefore, independent of each other in the encoding. Based on Table 3.4, parameters of a sparse lookup table can be encoded as

$$P_{\text{(Sparse Lookup Table)}} = [T_{int}, N_{sec}, N_{seg}] \tag{3.44}$$

Therefore, a real value encoded individual of the display rendering application can be elaborated as

$$\text{Individual } I_{\text{(Display Rendering)}} = [P^{\text{(Tone Mapping)}}_{\text{(Sparse Lookup Table)}}, P^{\text{(Color Space Conversion)}}, P^{\text{(EOTF Compensation)}}_{\text{(Sparse Lookup Table)}}]. \tag{3.45}$$

A typical example of a real-value encoded individual from the display rendering case study based on Table 3.2 and Table 3.4 is given as

$$\begin{aligned}
I_{\text{(Display Rendering)}} = \quad & [[1, 2, [256, 512]], && \rightarrow \text{ Tone mapping}\\
& \left.\begin{aligned}&[[[3, 7, 2], [[2, 2, 6], [2, 4, 11], [0]], 1, [[2, 5, 1], [0]]],\\
&[[3, 8, 0], [[0], [2, 1, 5], [0]], 0, [[4, 3], [2, 9, 0]]],\\
&[8, 2, 1], [[2, 5, 8], [0], [0]], 2, [[4, 5], [2, 1, 1]]]],\end{aligned}\right\} && \text{Color Space Conversion}\\
& [0, 8, [2, 16, 16, 16, 16, 32, 2, 512]]] && \rightarrow \text{ EOTF compensation}
\end{aligned}$$

For an easier real-value representation, we encode interpolation type T_{int} of the sparse tables as *None* = 0 and *Linear* = 1.

3.7.2.5 Genetic Operations

AxCGA performs genetic operations such as crossover and mutation independently on the sublist of parameters corresponding to each processing stage. Similar to the RGB to $YCbCr$ conversion case study, AxCGA selects random individuals from a parent population to perform these operations based on specified mutation and crossover probabilities.

For a mutation operation, AxCGA initially select one of the processing pipeline stages randomly, unlike in RGB to $YCbCr$ conversion which has a single processing stage. However, parameters of a random node from the selected stage is replaced afterward with another random choice within its parameter range. This parameter can be selected directly on stage level for the sparse lookup tables since only one module exists. In the color space conversion, AxCGA has to select one among three mixer modules within the stage to choose a random node. Following the parameter replacement of the selected node, a forward dependency propagation of parameters within the modules has to be taken care. Such a stage based mutation ensures modularity and generality of the operation. Together with a mutation rate mechanism in the adaptive GA approach, this generic mutation operation can be effectively adapted to multiple applications with different complexity.

The crossover operation is also performed stage basis to ensure modularity, and all the processing stages in an application pipeline undergo the crossover operation independently at the same time. A channel-based single-point crossover operation is performed for the color space conversion stage, as shown in Figure 3.13. However, a classical crossover is directly infeasible for sparse lookup tables in a tone mapping or EOTF compensation due to the differences in the length of individuals in a population. Depending on the number of sections N_{sec}, the number of sub-segments N_{seg} might differ for the selected parents for crossover operation. Therefore, we either upsample or downsample the parents to a same number of sections N_{sec} in order to keep same length for a classical crossover operation. These sampling operations are jointly developed with our post-doctoral researcher Anh Vu Doan. Figure 3.19 shows an example of such upsample or downsample operation in the sparse lookup table. An upsampling operation is straightforward, where a parent individual with 2 sections of 256 and 512 sub-segments can be upsampled to 4 sections with sub-segments 128, 128, 256, and 256. During the upsampling, it is essential to make sure that all the sub-segments must be greater than 1 to meet the parameter constraints. The downsampling operation needs to be performed carefully to create feasible design points. The same parent can be downsampled into 1 section and 768 sub-segments. Additionally, the downsampled sub-segments are required to round to a value of the next higher power of 2 to meet the parameter constraints, which is 1024 in this example. Therefore, among the selected parents for crossover, one of the individuals is either upsampled or downsampled to the same size as the second parent. Thereafter, a classical-single point crossover operation can be performed on these parents.

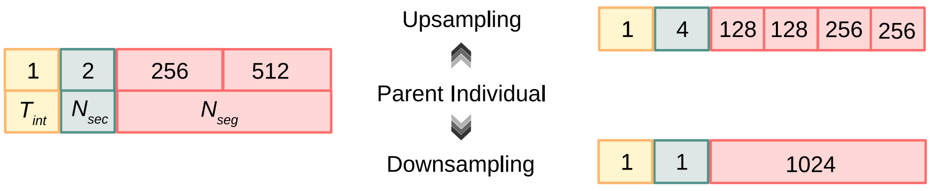


Figure 3.19: Upsampling and downsampling operations for crossover

3.7.2.6 DSE Objectives

The DSE objectives are defined for determining the trade-off between power consumption and quality degradation. Similar to the RGB to $YCbCr$ case study, the power is estimated directly from the area consumption. For quality models, we used CIELAB ΔE since the display rendering application deals with color transformation, which is often evaluated with human visual perception. We used both maximum ΔE and mean ΔE calculated across the training set for the quality estimation. The maximum ΔE estimates the worst-case error introduced by the approximations, which is of the highest interest to a designer to bind quality degradation reliably to the quality-threshold. The mean ΔE reflects an overall error in the training set and is usually not a main target of a designer. During the DSE, multiple configurations may have same worst-case maximum ΔE. In such cases, mean ΔE aids the AxCGA to differentiate these solutions and help to select a better configuration. Thereby, mean ΔE guides AxCGA to make finer improvements towards the optimum. Since ΔE is an error metric, the quality objective has to be minimized during the DSE. Therefore, the objective function in the display rendering application is defined as

$$f_{obj} = minimize\big(\text{maximum}(\Delta E), \text{mean}(\Delta E), \text{power}\big). \tag{3.46}$$

3.7.2.7 AxCGA Initialization and Setup

To start DSE experiments in AxCGA, both application-specific and GA-specific initialization have to be performed. During the application-specific initialization, we first created an application DFG of the display rendering pipeline. Thereafter, we estimated average toggle rate for the power model by simulating the reference design using the ARRI image set, resulting between 0.22 to 0.33. Therefore, we selected 0.28 as an average toggle rate in our AxCGA experiments. Similarly, per-unit power consumption is estimated as an average value for each FPGA resource from the reference implementation and 3 configurations with different resource consumption selected from 10 random approximate configurations. Finally, ΔE is selected as quality metrics together with relevant training data.

Since the display rendering pipeline is a color processing pipeline, a maximum worst-case error can be guaranteed only when all the possible input colors are used in a training dataset. Such a three-channel input color space for 12 bits per channel contains $2^{36} = 6.872 \times 10^{10}$ possible colors, which makes error estimation over the entire input space prohibitively long during the DSE. Therefore, we use a representative subset of the entire color space by sampling the color space in 128 steps in each direction. This reduces the training dataset to $128^3 = 2.097 \times 10^6$ different colors. The 128 step sampling uses the 7 bits in the MSB part of each pixel, and the remaining bits are filled with uniformly distributed noise to minimize the potential effect of a uniform sampling in the DSE. Figure 3.20 shows the resulting training image, and using such a reduced training dataset significantly reduces the quality estimation time at the expense of the accuracy of quality estimation. However, the training image still covers the entire color space.

The GA-specific initialization in the display rendering case study is similar to the initialization in the RGB to $YCbCr$ case study. We selected an adaptive version of the AxCGA with $\mu = 50$ and $\lambda = 100$, and group size parameter σ is selected as 5/8. The initial population includes a single

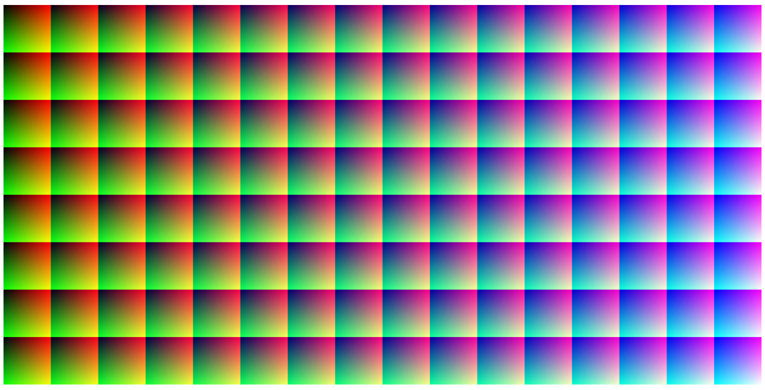


Figure 3.20: Training data with 128 steps used for quality estimation

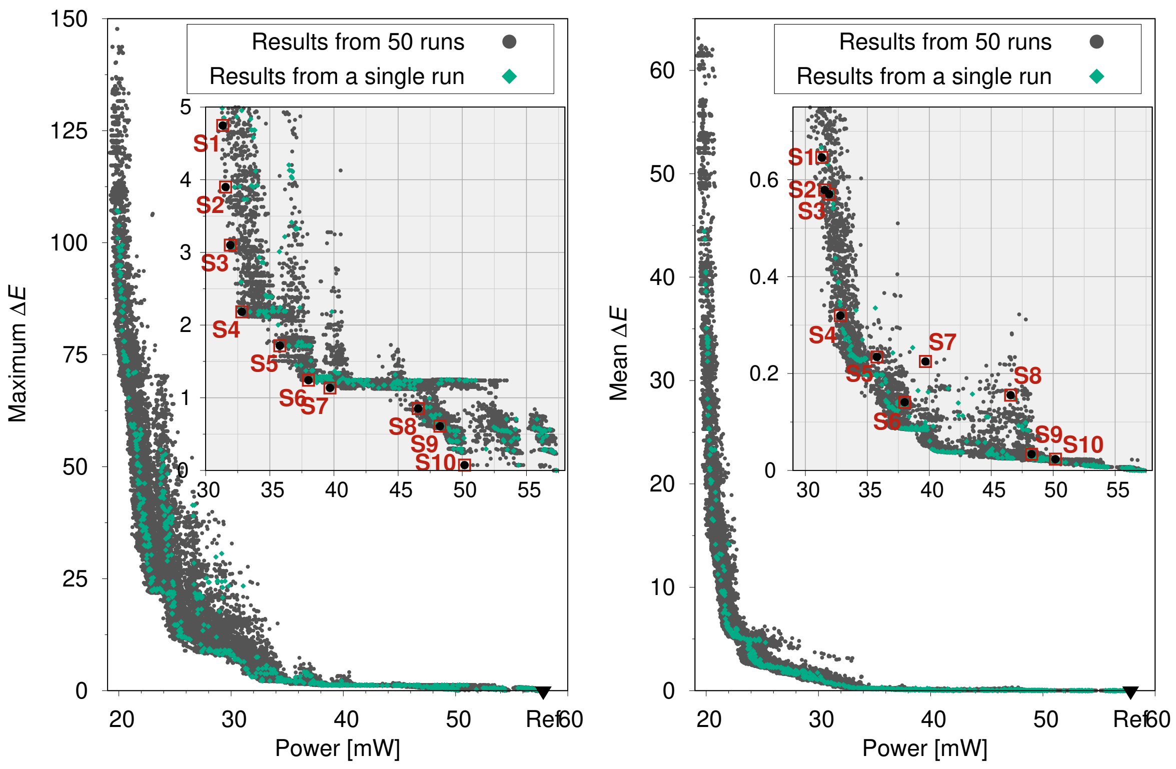
configuration corresponding to the reference implementation, and the remaining individuals are generated with a random selection of approximation parameters from its feasible ranges. Since the complexity of the design space is larger than the previous case study, we chose a higher number of generations N_{gen} as the termination condition. Based on our initial observations of the AxCGA performance, we selected $N_{gen} = 1000$ in all of our DSE experiments on the display rendering case study.

3.7.2.8 DSE Results

This section describes the DSE results obtained on the display rendering case study using AxCGA. The Pareto-optimal points from multiple experiments are discussed initially, and validation of our AxCGA approach using a set of selected configurations from one of the DSE results is performed following.

Pareto-optimal Solutions

We performed DSE using AxCGA on the display rendering case study and repeated the experiments 50 times independently to ensure the consistency of the results. The number of resulted Pareto-optimal solutions from each experiment varied from 456 to 707, averaging 565.38 solutions per experiment. Combined Pareto fronts from all these experiments are depicted as two-dimensional projections in Figure 3.21, where Figure 3.21a shows the most relevant trade-off between power consumption and the worst-case maximum ΔE, and Figure 3.21b shows the mean ΔE - power trade-off. Additionally, the points from a randomly selected experiment are highlighted as a representative result to demonstrate the trend of a single run. In both plots, all the irrelevant Pareto-optimal points that have higher power than the reference power due to approximation overhead have been excluded. From the figures, it can be seen that the points with maximum ΔE greater than 25 or mean ΔE greater than 7 do not significantly improve the power savings. Similarly, for configurations having more than 32 mW power, the mean or maximum error also does not improve significantly.



(a) Maximum ΔE vs. Power.

(b) Mean ΔE vs. Power.

Figure 3.21: 2D projections of Pareto fronts obtained from the 50 independent DSE experiments on display rendering application, and one randomly selected DSE front is highlighted. The inset shows the region of interest, and the reference design without any approximation is indicated as Ref.

DSE Results Validation

The perceiving color difference of humans lies at approximately 2.15 ΔE [241], and a useful trade-off region in practical applications lies around this threshold value. Therefore, we picked 10 points around this threshold value from a randomly selected experimental result to analyze the quality-resource trade-off further. We used maximum ΔE to select these points from a region between 5 and 0 since it reflects the worst-case error, keeping the mean ΔE below the threshold. Thereafter, we synthesized these configurations and estimated resources such as power, area, and speed using Intel Quartus Prime software. Additionally, we tested these configurations on all 12 images in the ARRI image set for quality validation. Table 3.5 lists the validation results of the selected configurations in terms of power, quality, utilization of each FPGA resource type, and speed. Even though the average error between the post-synthesis and model estimated power is 12.33% for the selected configurations, 100% fidelity guides the optimization to reach better points during the DSE. Overall, the post-synthesis power consumption of the selected configurations shows that the approximations save the power between 14.42% and 51.17% compared to 55.01 mW of the reference design power. The potential reduction in different FPGA resource types of each

Table 3.5: Resource usage, power consumption and quality data of the selected points

Selected	**Power**		**Quality**				**FPGA Resources**				**Speed**
Points	AxCGA	Synth.	Train	Test	Train	Test	DSP	LUT	Reg	BRAM	Synth.
[Index]	[mW]	[mW]	[Max ΔE]		[Mean ΔE]		[Units]	[Units]	[Units]	[Bits/Units]	[MHz]
S1	31.33	26.86	4.75	4.66	0.65	1.17	2	317	816	73 728/6	296.56
S2	31.56	27.73	3.90	3.72	0.58	1.04	2	329	827	73 728/6	294.81
S3	31.92	27.94	3.10	2.50	0.57	0.96	2	338	837	73 728/6	296.38
S4	32.82	29.31	2.18	2.32	0.32	0.47	2	344	862	73 728/6	295.25
S5	35.78	31.19	1.72	1.44	0.23	0.31	2	387	998	92 160/6	297.35
S6	38.01	33.15	1.24	1.30	0.14	0.20	4	369	1073	92 160/6	273.97
S7	39.69	35.36	1.14	1.25	0.22	0.36	4	398	1162	104 448/6	271.30
S8	46.56	41.45	0.85	0.84	0.16	0.21	3	384	1028	202 752/12	275.18
S9	48.26	45.50	0.61	0.58	0.03	0.10	5	338	1072	202 752/12	273.07
S10	50.18	47.08	0.07	0.07	0.02	0.02	5	368	1172	202 752/12	271.15
Ref	57.82	55.01	–	–	–	–	9	302	1068	294 912/18	269.83

selected configuration can also be seen in the table. Similarly, the speed obtained with the timing analyzer shows that all the configurations met the target frequency requirement of 266.66 MHz. Finally, quality validation using the test dataset shows 7.60% average error in maximum ΔE and 34.97% in mean ΔE. Even though the average error in mean ΔE from the quality validation is high compared to model estimation, the actual values from the quality validation are lower than the perceivable threshold value for all the configurations, which is critical in practical applications. Therefore, a designer can select configurations for the display rendering directly based on the trade-off between maximum ΔE and power.

3.8 Comparison between Adaptive and Non-Adaptive AxCGA

Real-world approximate computing application often exposes high design space complexity, and quality evaluation of each configuration using a training dataset usually consumes considerable time. In RGB to $YCbCr$ case study, the evaluation of one solution on an average takes approximately 84.47 milliseconds, whereas a single configuration evaluation on average in the display rendering application requires approximately 1.88 seconds on a workstation system (Intel Core i7-10700, 8 Cores, 32 GB RAM). Therefore, an extensive hyperparameter optimization with multiple parameter combinations in GA would be time-consuming and need to be repeated for every target design space. Addressing these issues in many state-of-the-art optimization problems, standard values are chosen for the GA parameters such as mutation probability p_m, crossover probability p_c, parent population size μ, and offspring population size λ, further denoted as non-adaptive approach. However, a better set of parameters often exists, even though the non-adaptive GA can produce Pareto fronts with many useful solutions. Overcoming the limitations of the non-adaptive approach, AxCGA uses a novel adaptive GA approach to provide hyperparameters adaptively during the DSE, as described in Section 3.6.4. The adaptive approach avoids the time-consuming hyperparameter optimization for individual applications and additionally provides an adequate set of hyperparameters for GA over generations.

To demonstrate the effectiveness of the adaptive approach, we compare the performance of the AxCGA with our novel adaptive GA hyperparameters and standard hyperparameter values used in the literature. We used the case studies introduced in previous sections for the experiments and performance comparison. In the non-adaptive approach, we selected p_c and p_m as commonly-used values such as 0.7 and 0.3, and μ and λ as 50 and 100, respectively [232]. The following experimental results show that the GA is able to produce Pareto fronts with many useful solutions in both scenarios.

In order to analyze the performance of these approaches, hypervolume indicator is estimated from all solutions identified in each generation of DSE. The hypervolume is a hybrid indicator measuring both convergence and diversity of a Pareto front [230]. Figure 3.22 illustrates the basic principle of a hypervolume computation in a minimization problem. The hypervolume indicator is estimated as the volume bounded by the solutions on a Pareto front and a reference point R. When an optimization progresses, the Pareto points might attain lower values, increasing the hypervolume values as well. Therefore, hypervolume values increase in general with a better optimization process.

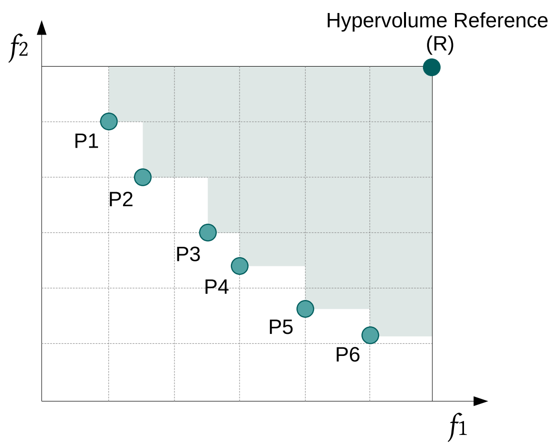
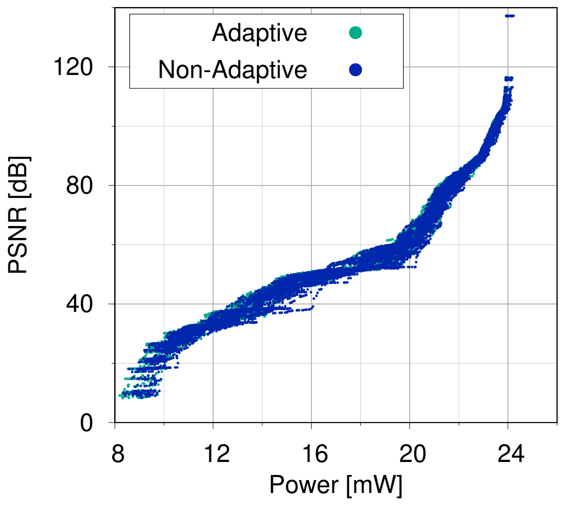


Figure 3.22: Example of the hypervolume indicator which is represented by the shaded area for Pareto points P1 to P6 with respect to the reference point R

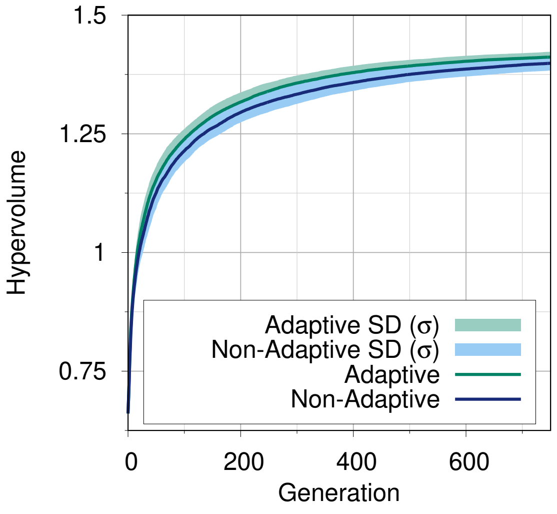
We repeated the DSE experiments using the non-adaptive hyperparameters 50 times independently for both RGB to $YCbCr$ conversion and display rendering case study. For the adaptive AxCGA results, we used previously described results in Section 3.7. The following sections show the Pareto solutions obtained and the average hypervolume values over the generations using both the adaptive and non-adaptive approaches from each case study.

3.8.1 Case Study 1: *RGB* to *YCbCr* Color Space Conversion

Figure 3.23a shows Pareto fronts obtained from 50 experiments of adaptive and non-adaptive AxCGA on the RGB to $YCbCr$ case study. Both approaches identified well-distributed Pareto curves that trade-off the application quality PSNR with system power. Further evaluating the performance of each approach quantitatively, we estimated the average hypervolume over the GA generations. The hypervolume reference for this case study is set experimentally to 7.78 and 29.67 mW for PSNR and power, which are 20% higher than the maximum identified values for each objective from multiple runs. The average hypervolume values over the GA generations are shown in Figure 3.23b. Overall, the adaptive approach performs slightly better with a final hypervolume of 1.4116 than the non-adaptive approach with 1.3986. Additionally, the standard deviation σ of the final hypervolume from the adaptive and non-adaptive approaches are 0.5835 and 0.5932, respectively. These values show comparable consistency of the results in both approaches, with a slightly better standard deviation value in the adaptive approach.



(a) Pareto fronts obtained from 50 experiments of adaptive and non-adaptive AxCGA

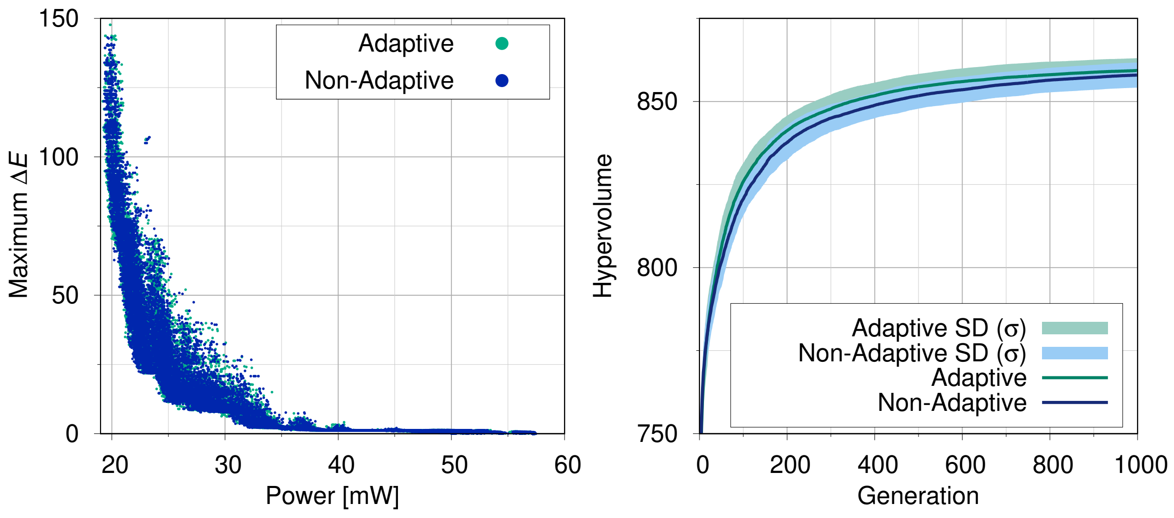


(b) Average hypervolume from 50 experiments of adaptive and non-adaptive AxCGA

Figure 3.23: Adaptive and non-adaptive AxCGA results on RGB to $YCbCr$ conversion

3.8.2 Case Study 2: Display Rendering Pipeline

Pareto trade-off curves in Figure 3.24a show results from 50 experiments of both adaptive and non-adaptive approaches on the display rendering pipeline. Since the maximum ΔE is of primary interest to a designer, we plot only the trade-off between maximum ΔE and power. From the figure, it is evident that both approaches are closely comparable and produce well-distributed solutions. Similar to case study 1, we set hypervolume reference experimentally to 206.61, 69.41, and 81.98 mW for maximum ΔE, mean ΔE, and power, respectively, The average hypervolume in Figure 3.24b shows that the adaptive approach performs marginally better with a final hypervolume of 859.32 than the non-adaptive approach with 857.95. Additionally, the standard deviations of the final hypervolume are 4.44 and 4.38, respectively, which indicates that the non-adaptive approach has slightly better consistency in producing the results, even though these values are closely comparable. Further analysis of the elite metaparameter group in each AxCGA generation shows that the hyperparameter values taken from the literature for the non-adaptive AxCGA are near-optimal, resulting in comparable performance in both versions of AxCGA.



(a) Pareto fronts obtained from 50 experiments of adaptive and non-adaptive AxCGA

(b) Average hypervolume from 50 experiments of adaptive and non-adaptive AxCGA

Figure 3.24: Adaptive and non-adaptive AxCGA results on display rendering application

3.9 AxCGA and autoAx DSE Comparison

The autoAx proposed for ASIC design [209] and ApproxFPGAs [211], an extension of autoAx approach to an FPGA platform, effectively combine multiple circuit level approximations from state-of-the-art approximate computing libraries in an application. Due to architectural differences in these target platforms, resource modeling approaches are distinct in both approaches to ensure good fidelity. However, both approaches use a similar technique inspired from a hill climbing algorithm to construct Pareto-optimal solutions. This DSE approach for Pareto construction was originally proposed in autoAx and later used in ApproxFPGAs as AutoAx-FPGA. Therefore, we denote the Pareto-optimal construction algorithm as autoAx DSE. We performed multiple DSE experiments using autoAx DSE approach for RGB to $YCbCr$ conversion and display rendering application and compared the performance with the AxCGA results. The following section describes the principle of Pareto-optimal construction in autoAx, explains how the autoAx is employed to approximate the case studies, and compares the autoAx DSE results with the AxCGA results.

3.9.1 autoAx DSE

The autoAx approach consists of three design phases: a preprocessing phase of approximate arithmetic circuits, a quality and resource modeling phase, and an iterative DSE phase identifying

Pareto-optimal solutions. In the preprocessing phase, characterization of multiple state-of-the-art approximate arithmetic components using benchmark data is performed in autoAx. As a result, a subset of the components which offers Pareto trade-off between characterization error and hardware resources is stored in a library. Thereafter, a set of configurations for a target application is generated by randomly replacing accurate components with their approximate counterparts from the library. These random configurations are synthesized and simulated for hardware resources and quality, and machine learning models are trained independently for hardware resources M_{HW} and quality of results M_{QoR}.

The principle of autoAx DSE which iteratively forms Pareto-optimal solutions is explained in Algorithm 15. The inputs to the autoAx DSE are a library of parametrizable approximate components RL and models for hardware cost estimation M_{HW} and quality of results M_{QoR}. Initially, a parent configuration is chosen by replacing accurate circuit components in a target application with random approximate counterparts from the preprocessed library, and an empty Pareto front P is initialized. Thereafter, a neighbor configuration C is derived from this parent by modifying some components or parameters of the parent. Then, evaluate the fitness of the neighbor configuration C using pre-trained models for the quality e_{QoR} and hardware cost e_{HW}. If the newly evaluated neighbor can be a Pareto point in P, this point will be added to P. In addition, other existing points in P that become non-Pareto points due to this new point will be removed from P. In this case, the neighbor configuration acts as the new parent in the next autoAx iteration. However, if the neighbor is not a Pareto point in the P, a new neighbor is formed from the same parent by replacing different components or parameters. If a stagnation condition is detected such that the neighbor configurations formed from a parent cannot be a Pareto point for a certain number of DSE iterations, the parent is replaced with one of the random configurations already added to P. This process will iterate until the termination condition occurs, and finally, the Pareto-optimal configurations are returned.

3.9.2 Comparison between autoAx DSE and AxCGA

We used autoAx DSE approach to combine multiple approximations and find the Pareto-optimal solutions in both the RGB to $YCbCr$ color space conversion and display rendering application with the same experimental setup described in Section 3.7. The preprocessed approximate library in the autoAx is replaced with the approximate computing library used in AxCGA, which includes approximation methods in multi-level abstractions. Synthesizing multiple random configurations of our case studies is very time-consuming, where a single configuration requires 2.38 minutes in RGB to $YCbCr$ conversion and 2.48 minutes in display rendering application on a workstation system (Intel Core i7-10700, 8 Cores, 32 GB RAM). Therefore, autoAx models which require multiple synthesis results to train machine learning models are also replaced with the models proposed in AxCGA. The representation of a typical parameter configuration in autoAx is similar to the parameter configuration in AxCGA due to its real value genetic encoding. However, autoAx requires an additional `getNeighbor()` function which modifies parameters of an autoAx parent to get a suitable neighbor configuration. The below sections describe different neighbor functions used in autoAx approach for each case study and compare the DSE results with our AxCGA approach. The

Algorithm 2: Pareto set construction in autoAx [209]

```
Input  : $RL$ - set of libraries, $RL = \{RL_1, RL_2, \dots, RL_n\}$,
         $M_{HW}$ - HW costs model, $M_{QoR}$ - quality model
Output : Pareto set $P \subseteq RL_1 \times RL_2 \times \dots \times RL_n$
1  function HEURISTICPARETOCONSTRUCTION($RL, M_{QoR}, MC$):
       /* initialization of autoAx                                  */
2      $Parent \leftarrow$ PICKRANDOMLYFROM($RL_1, RL_2, \dots, RL_n$) ;
3      $P \leftarrow \emptyset$ ;
       /* main loop represents iterations in autoAx                 */
4      while TerminationCondition do
5          $C \leftarrow$ GETNEIGHBOR($Parent$) ;
6          $e_{QoR} \leftarrow M_{QoR}(C)$ ;
7          $e_{HW} \leftarrow M_{HW}(C)$ ;
8          if PARETOINSERT($P, (e_{QoR}, e_{HW}), C$) then
9              $Parent \leftarrow C$;
           /* parent is not changed in last $k$ iterations              */
10         else if StagnationDetected then
11             $Parent \leftarrow$ PICKRANDOMLYFROM($P$) ;
12         end
13     end
14     return $P$ ;
15     end function ;
```

stagnation condition in autoAx experiments is chosen as 5 iterations without any improvement from a parent based on a few preliminary trials.

3.9.2.1 Case Study 1: *RGB* to *YCbCr* Color Space Conversion

The possible approximation components and their parameter ranges in RGB to $YCbCr$ conversion are the same as in Table 3.2. A typical parent in autoAx is similar to an individual in AxCGA and can be represented as

$$\text{Parent}_{\text{(RGB to YCbCr)}} = [P^{\text{(RGB to YCbCr)}}] \tag{3.47}$$

$$= [P^1_{mixer}, P^2_{mixer}, P^3_{mixer}], \tag{3.48}$$

$$\text{where } P_{mixer} = [[F^1_{\text{co}}, F^2_{\text{co}}, F^3_{\text{co}}], [M^1, M^2, M^3], F_{\text{in}}, [A^1, A^2]]. \tag{3.49}$$

Similarly, the objective function is defined as

$$f_{obj} = minimize\big(\text{power}\big) \;\wedge\; maximize(\text{PSNR})\,. \tag{3.50}$$

In the RGB to $YCbCr$ conversion, we introduced four different neighbor functions based on possible parameter choices of a typical parent.

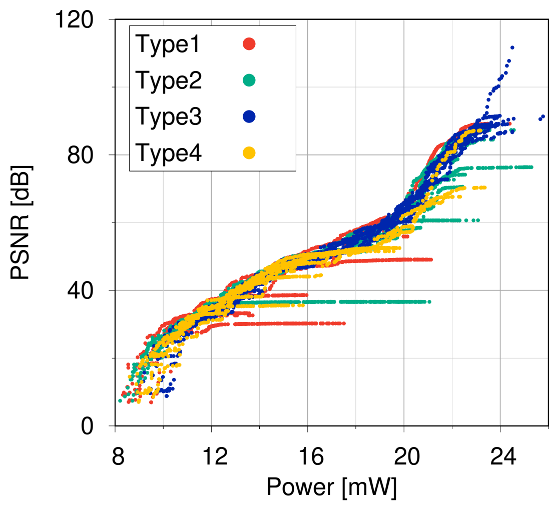
- Type 1: Single Sweep - choose one parameter from a parent and replace it with a random choice from its possible parameter range. Additionally, if the new parameter influences any other parameters in the parent due to parameter dependencies within a module, these parameters should also be adjusted accordingly, similar to AxCGA mutation operation.

- Type 2: Double Sweep - performs two Single Sweep operations successively by considering design constraints and parameter dependencies within a module.
- Type 3: Mixed Mixer Sweep - either performs a Single Sweep operation on a parent or randomly regenerates parameters of one of the three mixer channels in the parent with equal probabilities.
- Type 4: Full Sweep - choose one parameter from each mixer module in a parent and replace it with a random choice from its possible parameter range. Similar to a Single Sweep, all the dependent parameters are also modified by maintaining the design constraints.

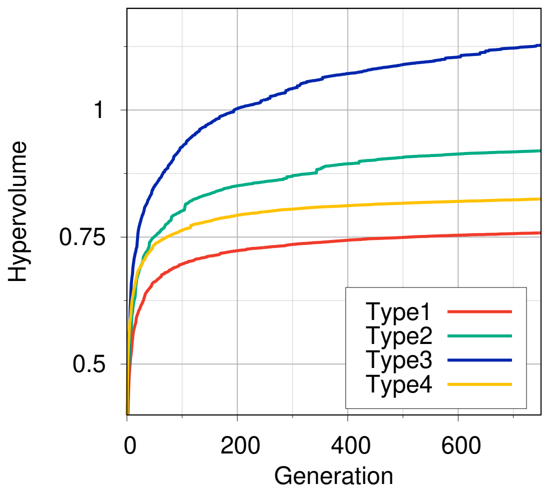
Using these neighbor functions, we performed autoAx DSE 10 times independently to identify the best neighbor function that modifies the parent effectively. Similar to the AxCGA experiments, the same 64^3 pixels training dataset is used in all experiments in RGB to $YCbCr$ conversion. A single generation in AxCGA is equivalent to the 100 autoAx DSE iterations due to a similar number of fitness evaluations. Therefore, a termination condition is chosen as 75 050 fitness evaluation in autoAx, which is equivalent to 750 generations in AxCGA. To compare the performance of Pareto fronts evolved with each neighbor function, we use hypervolume values with the same reference points estimated in Section 3.8.1. We computed average hypervolume at every 100 iterations in autoAx to align with AxCGA results and make performance comparison easier.

Figure 3.25a shows combined Pareto-optimal solutions trading off PSNR and power from 10 independent experiments using four different neighbor functions. It can be seen that both Type 1 and Type 2 functions are stuck at local optima in different regions, resulting in points aligning horizontally with the same PSNR values for different power consumption values. Among the four types, only the Type 3 Mixed Mixer Sweep function finds points near the maximum PSNR quality region. The average hypervolume values from the 10 runs are depicted in Figure 3.25b, and it can be seen that Type 3 performs substantially better than all other neighbor functions. Therefore, we selected autoAx with Type 3 Mixed Mixer Sweep neighbor function for a performance comparison with AxCGA.

We additionally repeated autoAx DSE experiments with the Type 3 neighbor function 40 more times and compared the overall results from 50 independent experiments with AxCGA. The combined Pareto points from both approaches in Figure 3.26a demonstrate the PSNR-power trade-off. Analyzing the trade-off further, it is evident that both approaches are performed competitively in the middle region. However, due to the inherent diversity and convergence property of NSGA-II selection, points on the extremes of the trade-off space are identified better in AxCGA. The average hypervolume values for both approaches in Figure 3.26b show that the final hypervolume obtained with autoAx experiments can be achieved in AxCGA within 50 generations of AxCGA.

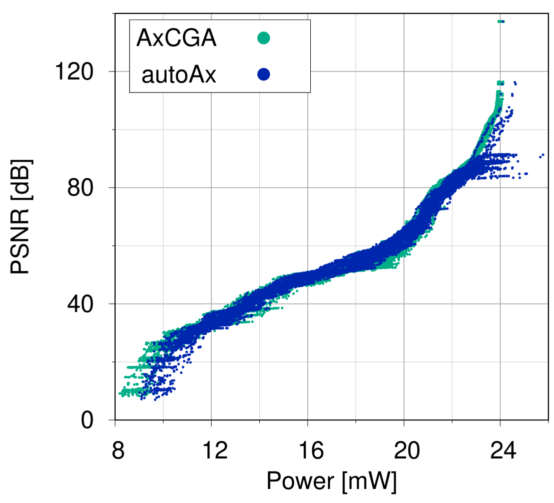


(a) PSNR[dB] vs. Power.

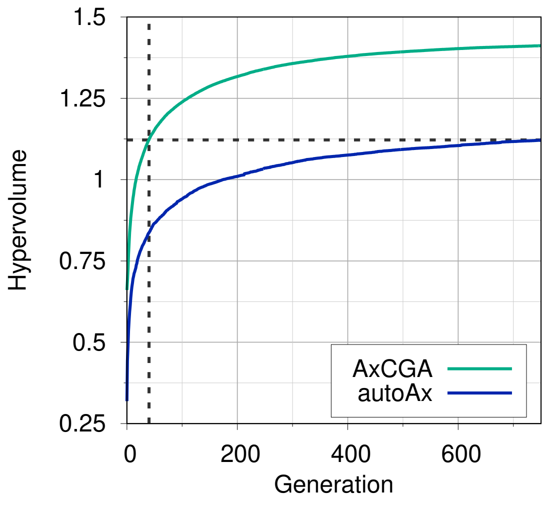


(b) Hypervolume over DSE generations. A single AxCGA generation is equivalent to 100 iterations in autoAx

Figure 3.25: autoAx DSE results obtained for different neighbor functions from the 10 independent experiments on RGB to $YCbCr$ conversion



(a) PSNR [dB] vs. Power.



(b) Hypervolume over DSE generations. A single AxCGA generation is equivalent to 100 iterations in autoAx

Figure 3.26: DSE results obtained from the 50 independent experiments on RGB to $YCbCr$ conversion

3.9.2.2 Case Study 2: Display Rendering Pipeline

We performed autoAx DSE on display rendering application with the same experimental setup described in Section 3.7.2. A typical parent includes approximation parameters from a tone mapping stage, a channel mixer stage, and an EOTF compensation stage. Therefore, a parent in autoAx is represented as

$$\text{Parent}_{\text{(Display Rendering)}} = [P^{\text{(Tone Mapping)}}_{\text{(Sparse Lookup Table)}}, P^{\text{(Color Space Conversion)}}, P^{\text{(EOTF Compensation)}}_{\text{(Sparse Lookup Table)}}], \text{ where} \tag{3.51}$$

$$P_{\text{(Sparse Lookup Table)}} = [T_{int}, N_{sec}, N_{seg}] \text{ and } P_{\text{(Color Space Conversion)}} = [P^1_{mixer}, P^2_{mixer}, P^3_{mixer}]. \tag{3.52}$$

Each of these parameters can vary according to ranges specified in Table 3.2 and Table 3.4.

The DSE objective function is defined as

$$f_{obj} = minimize\big(\text{maximum}(\Delta E), \text{mean}(\Delta E), \text{power}\big). \tag{3.53}$$

Due to an increased design space complexity compared to the RGB to $YCbCr$ conversion, we considered seven different neighbor functions that randomly modify a parent in our preliminary experiments.

- Type 1: Single Sweep - modify one parameter and its dependencies (cp. Section 3.9.2.1)
- Type 2: Double Sweep - modify two parameters and their dependencies successively (cp. Section 3.9.2.1)
- Type 3: Single Sparse-Single Mixer Sweep - modifies one parameter and its dependencies from two sparse lookup tables and one parameter and its dependencies from the color space conversion stage.
- Type 4: Single Sparse-Double Mixer Sweep - modifies one parameter and its dependencies from two sparse lookup tables and two parameters and their dependencies successively from the color space conversion stage.
- Type 5: Independent Sweep - modifies one parameter each from tone mapping, color space conversion, and EOTF compensation stages by maintaining the dependencies within each module.
- Type 6: Single Sparse-Mixed Mixer Sweep - modifies one parameter and its dependencies from two sparse lookup tables and performs a Mixed Mixer Sweep (cp. Section 3.9.2.1) in the color space conversion stage.
- Type 7: Single Sparse-Full Sweep - modifies one parameter and its dependencies from two sparse lookup tables and performs a Full Sweep (cp. Section 3.9.2.1) in the color space conversion stage.

We independently ran autoAx DSE experiments 10 times using each neighbor function to identify the best-performing function. The termination condition is chosen as 100 050 iterations, which is equivalent to 1000 generations in AxCGA. Similar to the RGB to $YCbCr$ case study, we computed hypervolume values every 100 iterations in autoAx DSE to reduce computational time and ease the performance comparison with AxCGA. The time required to perform an autoAx DSE experiment using the 128-step training image is measured in days due to sequential fitness evaluations in autoAx DSE. Therefore, we used a reduced 16-steps image shown in Figure 3.27 in the neighbor search experiments, which contains 16^3 pixels obtained by sampling color space in 16 steps in each dimension.

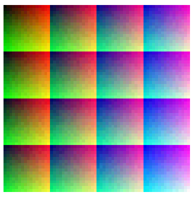
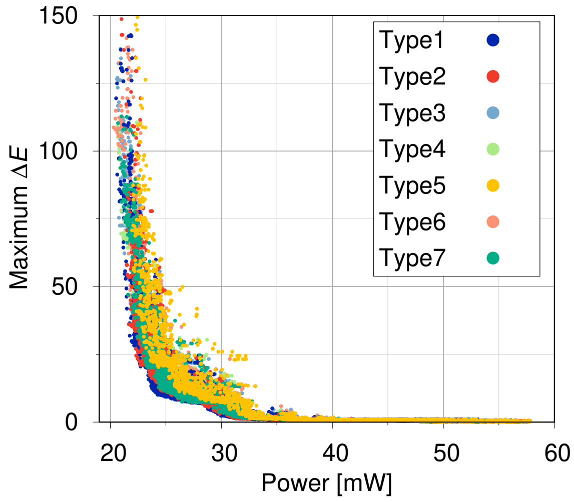
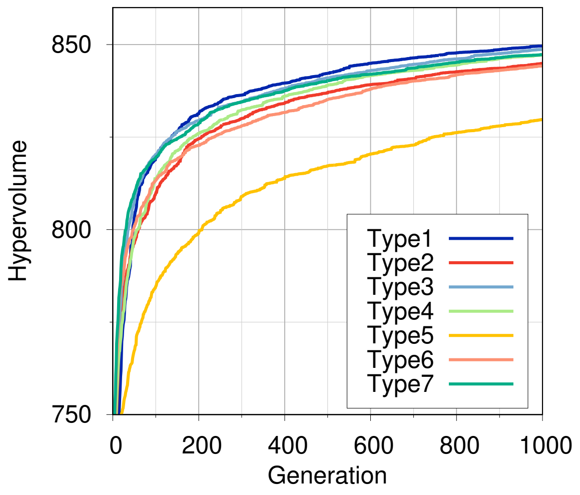


Figure 3.27: Training data with 16 steps used in autoAx neighbor search

Figure 3.28a depicts the solutions that trade-off the worst case maximum ΔE and power from 10 independent experiments of autoAx, and it can be seen that all neighbor functions comparably identified the trade-off. Further analyzing the quality of Pareto fronts obtained from each neighbor function type, we plotted average hypervolume over generations in Figure 3.28b. From the figure, it can be seen that the Type 1 Single Sweep neighbor function performed marginally better than the other neighbor functions, whereas the Type 5 Independent Sweep performed significantly worse compared to other neighbor functions. Therefore, we use the Single Sweep neighbor function for the performance comparison of autoAx DSE with AxCGA.



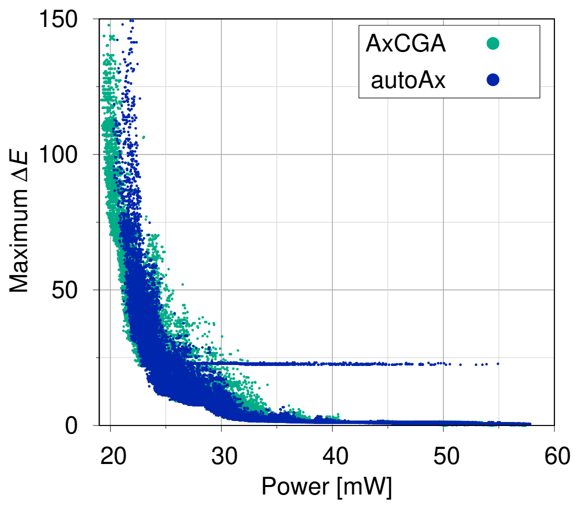
(a) Maximum ΔE vs. Power.



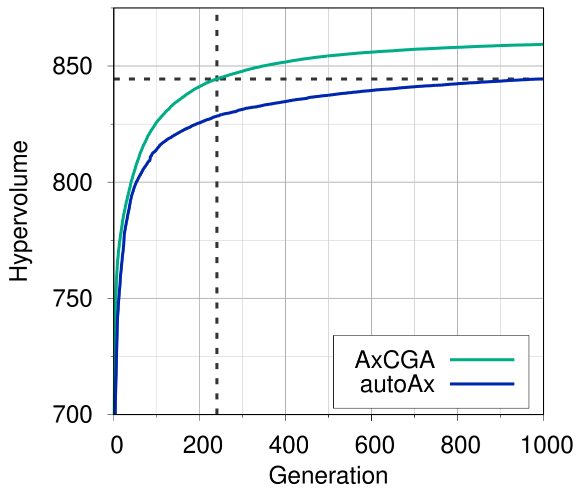
(b) Hypervolume over DSE generations. A single AxCGA generation is equivalent to 100 iterations in autoAx.

Figure 3.28: autoAx DSE results obtained for different neighbor functions from the 10 independent experiments on display rendering application

For performance comparison, we ran the autoAx DSE independently with the Single Sweep neighbor function using 128-steps images 50 times. Figure 3.29a shows a two-dimensional projection of solutions that trades off maximum ΔE and power obtained from all 50 experiments of autoAx DSE and AxCGA. Both approaches identified solutions comparably in the knee of the curves. However, within a region above 50 maximum ΔE, the AxCGA performs significantly better than the autoAx. In addition, it can be seen that one autoAx DSE is stuck in a local optimum without improving the quality further, leading to many solutions aligned horizontally. The average hypervolume over the DSE generations in Figure 3.29b shows that a final hypervolume achieved using autoAx can be obtained within 240 generations of AxCGA approach.



(a) Maximum ΔE vs. Power.



(b) Hypervolume over DSE generations. A single AxCGA generation is equivalent to 100 iterations in autoAx

Figure 3.29: DSE results obtained from the 50 independent experiments on display rendering application

3.9.2.3 Runtime Complexity Comparison

Similar to AxCGA, the autoAx DSE runtime can also break down into two components: the time required for all the fitness evaluations and algorithmic executions.

$$t_{autoAx} \approx time\ for\ fitness\ evaluations + time\ for\ algorithmic\ execution \tag{3.54}$$

The overall time for fitness evaluations in autoAx is a product of the number of fitness evaluations and the average fitness evaluation time due to a sequential execution. Since we have the same number of fitness evaluations in both autoAx DSE and AxCGA experiments, the number of autoAx fitness evaluations can be derived as $(\mu + \lambda \times N_{gen})$ from Equation 3.22. The algorithmic execution time t_a of autoAx mainly includes the time required to generate neighbor functions and check the neighbor point for the Pareto property in every iteration. Therefore, the total DSE time in autoAx can be estimated as

$$t_{autoAx} \approx [(\mu + \lambda \times N_{gen}) \times t_f] + [(\mu + \lambda \times N_{gen}) \times t_a]. \tag{3.55}$$

However, time for algorithmic execution is negligible due to the dominance of fitness evaluation time in real-world image processing applications. Therefore, the overall time is approximated as

$$t_{autoAx} \approx (\mu + \lambda \times N_{gen}) \times t_f. \tag{3.56}$$

Upon comparing t_{autoAx} with t_{AxCGA} in Equation 3.24, the AxCGA experiments can be sped up approximately by a factor of $min(N_{cores}, \lambda)$.

We avoid evaluating redundant configurations in a single DSE experiment in both approaches by maintaining a history of fitness evaluations. This further speeds up the DSE depending on the number of redundant configurations generated. Up on analyzing time from random DSE experiments, we observed that a single AxCGA experiment required 27.22 minutes, whereas autoAx took 1.49 hours in RGB to $YCbCr$ conversion on a workstation system (Intel Core i7-10700, 8 Cores, 32 GB RAM). Similarly, a single AxCGA required 6.96 hours and autoAx DSE needed 1.49 days in the display rendering case study.

3.10 Summary

This chapter addresses the problem of optimally configuring parameters for FPGA-based approximate image processing systems. We proposed a methodology named AxCGA for determining quality-resource trade-off using multi-objective optimization, which can explore complex design spaces exposed from combined single-purpose approximations on multiple abstraction levels. We employed fast, simple, yet accurate models to determine the fitness of each configuration probed during the AxCGA experiments. For a multi-objective DSE, we used a GA-based metaheuristic, together with NSGA-II selection which globally search a design space by maintaining both the convergence and diversity in the identified solutions. This chapter also describes in detail how DFGs are formed in an approximation application and how the genetic operations are performed by considering parameter dependencies and design constraints. A novel adaptive GA approach supplies hyperparameters during the DSE in AxCGA, avoids time-consuming hyperparameter optimization, and overcomes potential limitations that can occur from the standard hyperparameters from the literature. We demonstrate the generality and reusability of AxCGA in two real-world applications, such as RGB to $YCbCr$ color space conversion and display rendering application, where the second application reuses the approximation stage from the first application. The experimental results show that the AxCGA successfully determined the quality-resource trade-off offered by these applications with different approximate computing techniques. Comparing the performance of AxCGA with state-of-the-art autoAx approach, AxCGA determined the Pareto-optimal solutions efficiently and effectively in both case studies.

CHAPTER 4

Region of Interest Based NSGA-II (ROI-NSGA-II) AxCGA

This chapter describes a novel ROI-NSGA-II selection algorithm which helps to concentrate the GA search pressure into an ROI in a design space and how the new ROI-NSGA-II improves DSE efficiency in AxCGA. The core methodology is initially presented in "Region of Interest-Based Parameter Optimization for Approximate Image Processing on FPGAs" @IJNC 2021 [24] and later extended in "Region of interest based non-dominated sorting genetic algorithm-II: an invite and conquer approach" @ACM 2022 [25]. Therefore, figures and texts used in this chapter are adapted and extended from these publications.

4.1 Introduction

Over the past decades, evolutionary computing has been gained broad research attention in solving real-world MOO problems. The proposed AxCGA identifies a well-distributed and converged set of Pareto solutions that trades in application quality for hardware resources using the inherent capabilities of the NSGA-II in maintaining both the convergence and divergence simultaneously. The AxCGA approach using NSGA-II selection is further denoted as NSGA-II AxCGA in the following sections. The experimental results on real-world applications shown in Chapter 3 demonstrate these capabilities, and the Pareto fronts are identified efficiently and effectively in these applications (cp. Figure 3.15 and Figure 3.21).

Upon further analyzing the Pareto fronts, the size of the useful trade-off region might vary depending on the application properties and designer requirements. In RGB to $YCbCr$ conversion, this region covers most of the design space, whereas, in display rendering application, this useful region is only very small in size compared to the entire design space. However, the classical NSGA-II-based DSE globally explores a solution space defined by the design constraints and spends equal efforts to optimize the entire design space. Therefore, in applications where a designer is interested in a subset of a design space, a global optimization wastes effort to find irrelevant solutions outside the desired region.

To concentrate DSE efforts into a certain region, multiple approaches have been proposed over the years [243, 244, 245] which incorporate a preference information before (a priori), after (a posteriori), and progressively (interactive) during the Non-dominated Sorting Genetic Algorithm (NSGA)-based optimizations [246, 247]. An a priori method allows a designer to specify the preference information before an optimization starts. After that, designer interaction is not permitted in such methods. However, using this preference information, such methods often restrict the search to a specific region, reducing computational effort and irrelevant fitness evaluations. The preference information is usually given as reference points [242, 247, 248, 249, 250, 251, 252], direction [253], weight vectors [254], or a combination of more than one of these types [255]. Providing such information often requires a good knowledge about the final optimality and strongly influences the explored region.

A promising NSGA-II variant Reference Point-Based Non-dominated Sorting Genetic Algorithm-II (R-NSGA-II) proposed by Deb and Sundar [242] explores a preferred region by modifying a crowding distance based selection to emphasize the optimization efforts into desired regions. A designer has to provide preference information as reference points and a distance variable epsilon ϵ, which controls the spread of solutions in a Pareto set during selection. The R-NSGA-II also starts similar to the classical NSGA-II in which initial points or individuals are selected based on their dominance, starting from the first non-domination level. However, from a final relevant non-domination level that might contain more individuals than a selection requires, the individuals are selected based on normalized Euclidean distance from the given reference points instead of classical crowding distance based selection. In R-NSGA-II, both the reference point and the epsilon value determine the optimization efforts within the desired region. Filatovas et al. extended the R-NSGA-II approach and proposed Synchronous R-NSGA-II that employs three scalarizing functions instead of the Euclidean distance [248]. By adjusting these scalarizing functions, the algorithm can simultaneously concentrate its search on several regions, introducing more flexibility in preference incorporation. To improve convergence and diversity of the selected solutions, Filatovas et al. later introduced another variant of R-NSGA-II by incorporating a heuristic local search strategy named multi-objective single-agent stochastic search [252]. Further extending the R-NSGA-II, Li et al. replaced the normalized Euclidean distance in R-NSGA-II with a Chebyshev distance during the selection of individuals from the final relevant non-domination level [256]. This approach can identify solutions in an objective space that are hard to reach by R-NSGA-II. Deb et al. introduced Reference Direction Based NSGA-II (RD-NSGA-II) that combines a reference direction approach with the classical NSGA-II [253]. A designer has to provide a starting point and a reference vector that defines the reference direction as preference information. Then, a non-dominated sorting is performed based on a scalarizing function for a set of points in the reference directions, and this approach also demonstrated its effectiveness on a set of benchmark problems. A hybrid approach presented by Deb et al. employs the principles of the light beam search in NSGA-II to solve multi-objective optimization problems with designer preference. A desirable point named aspiration point and a reservation point beyond which the objective function values are not admissible can be provided optionally. During the optimization, a reference direction or light beam is formed using these points. Thereafter, a part of the Pareto-optimal region illuminated by the light beam with a span controlled by a veto threshold is selected. This approach is also experimentally proven on different benchmark problems. Molina et al. introduced a new approach named g-dominance that modifies traditional Pareto dominance relations [251]. During the selection, solutions satisfying all or none of the aspiration levels are preferred over solutions satisfying some aspiration levels in this approach. Similarly, Said et al. proposed the r-dominance approach, which creates a strict partial order between non-dominated solutions considering their weighted Euclidean distances to a reference point [247].

Many of the above approaches in literature have already demonstrated their usability and potential with different test problems. The selection of a typical method in an application predominantly depends on the ease of providing preference information to explore a desired region and its efficiency in guiding the search pressure into this region. In many real-world optimization problems, such as approximate image processing, however, reference points, directions, or weights cannot be efficiently provided due to the lack of knowledge about final optimality. These established methods need additional effort or preliminary trials to provide this information accurately.

Overcoming the above limitations, we propose ROI-NSGA-II, a novel approach of incorporating the preference information to the NSGA-II. In contrast to other types of preference information, a designer can easily specify threshold values for each objective as preference information in many real-world applications. These threshold values can, therefore, directly define an ROI as shown in Figure 4.1. Our proposed ROI-NSGA-II incorporates the threshold values also into the search and concentrates the optimization efforts into the corresponding ROI in a three-phase operation, namely *initialization*, *invitation* and *conquest*. In approximate image processing applications, an ROI can be defined as a subset of the entire design space using an additional quality-threshold parameter (Q_{th}) introduced by approximate computing. By integrating this quality-threshold also into DSE with ROI-NSGA-II, the search pressure can be directly concentrated into the desired region, and this makes the DSE in AxCGA more computationally efficient. The AxCGA approach using our proposed ROI-NSGA-II selection is denoted as ROI-NSGA-II AxCGA in the following sections.

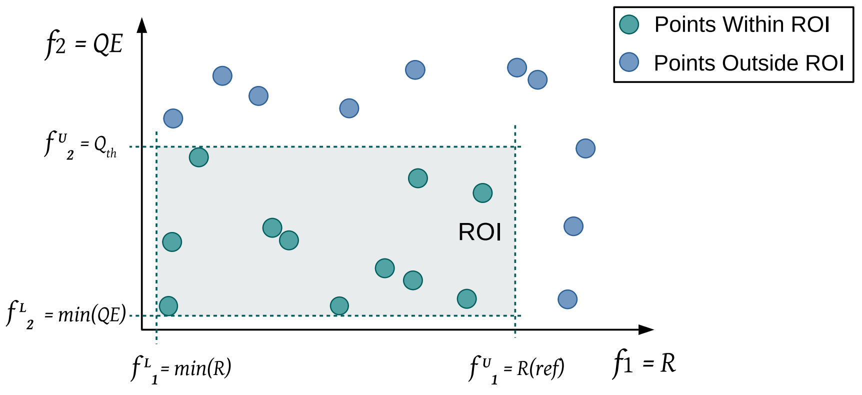


Figure 4.1: An ROI defined using the threshold values in an exemplary two objective minimization problem

The quality of an image processing application is often estimated as either the quality value Q directly, such as $PSNR$, or as a quality error (QE), such as ΔE or absolute error. Therefore, the type of quality-resource optimization problem varies depending on the selected quality metric in an application. The quality value Q leads to a maximization problem in which a designer is typically interested in a region between the maximum quality ($max(Q)$) and the quality-threshold value (Q_{th}). Whereas, quality error (QE) leads to the minimization of error values in which a region between minimum possible quality error ($min(QE)$) and quality-threshold value might be interesting for a designer, as shown in example Figure 4.1. Therefore, an ROI can be defined using the quality-threshold value (Q_{th}) as one boundary and keeping the other boundary with the minimum possible quality error ($min(QE)$) or maximum possible quality ($max(Q)$). Similarly, hardware resources, such as area or power values of a reference implementation ($R(ref)$), can be the upper threshold value in the resource dimension R. All irrelevant configurations with higher resource values than the reference implementation, due to the approximation overhead, are thereby excluded from the ROI. The lower boundaries of the resources R are kept at minimum possible resource utilization *min(R)*. This directly defines an ROI with $f^U_{1=R} = R(ref)$, $f^L_{1=R} = min(R)$, $f^U_{2=Q/QE} = max(Q)/Q_{th}$, and $f^L_{2=Q/QE} = Q_{th}/min(QE)$.

The incoming sections in this chapter are organized as follows. Section 4.2 describes three phases of our proposed ROI-NSGA-II methodology in detail. Thereafter, integration of the proposed ROI-NSGA-II to AxCGA framework is briefly described in Section 4.3. In Section 4.4, the performance of the ROI-NSGA-II AxCGA is demonstrated and compared with NSGA-II based approach and autoAx DSE. To further demonstrate the potential of adaptive GA, an additional comparison between the adaptive and non-adaptive version of AxCGA ROI-NSGA-II is included in Section 4.5. Finally, this chapter summarizes in Section 4.6.

4.2 ROI-NSGA-II Methodology

The proposed ROI-NSGA-II integrates boundaries of an ROI defined by a designer to the classical NSGA-II search. In contrast to many state-of-the-art approaches, the preference information can be directly supplied as the acceptable lower and upper threshold values for each objective without any preliminary trials. An overview of our ROI-NSGA-II methodology is shown in Figure 4.2. In general, the ROI-NSGA-II operates in three distinct phases, namely initialization, invitation, and conquest, and each phase is described in detail in the following sections.

4.2.1 Phase I: Initialization

The first initialization phase starts with classical NSGA-II and identifies points globally from the entire design space (cp. Section 3.6.3.3). During this phase, the designer-given preference information is not used for the search. The idea of the initialization phase is to use the inherent capabilities of NSGA-II in maintaining diversity and convergence in a selection process and identify the first set of points within the ROI. This initialization phase lasts until more than one point is identified within the ROI to make a meaningful selection using these points in the following invitation phase.

The duration of an initialization phase depends on the size and position of an ROI and the number of objectives. If the size of the ROI is considerably large compared to the entire design space, the probability of finding a solution early within the ROI increases using NSGA-II. In many

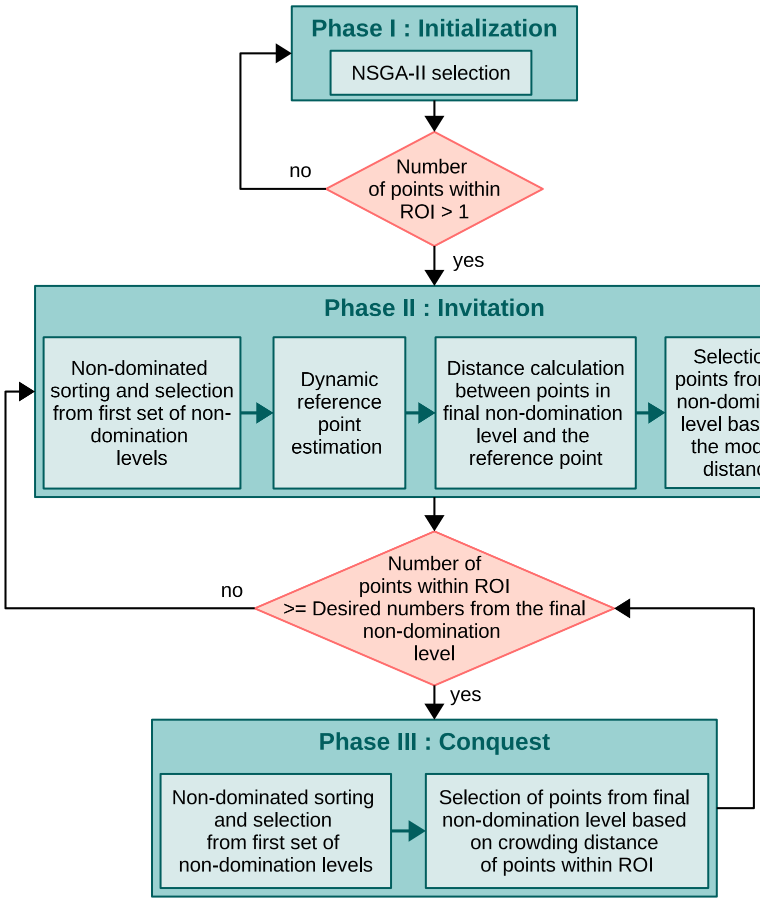

Phase I : Initialization
NSGA-II selection
Number of points within ROI > 1
no
yes
Phase II : Invitation
Non-dominated sorting and selection from first set of non-domination levels
Dynamic reference point estimation
Distance calculation between points in final non-domination level and the reference point
Selection of points from final non-domination level based on the modified distances
Number of points within ROI >= Desired numbers from the final non-domination level
no
yes
Phase III : Conquest
Non-dominated sorting and selection from first set of non-domination levels
Selection of points from final non-domination level based on crowding distance of points within ROI

Figure 4.2: Overview of the ROI-NSGA-II selection

cases, the NSGA-II identifies the first set of points within a few GA generations. However, suppose an ROI is located in a region close to a local optimum that optimization algorithms cannot reach easily. In that case, an initialization phase might require more generations to identify the first set of points within the desired ROI. Similarly, once the number of optimization objectives increases, the number of constraints increases due to additional threshold values in each objective. Therefore, finding the first set of points that satisfies all of these constraints might also take some additional generations.

Due to an initialization phase, the ROI-NSGA-II guarantees a minimum performance that an NSGA-II can deliver, even if the threshold boundaries set by a designer are unrealistic to an application. Since the primary purpose of the initialization phase is to identify the first set of points within an ROI, the initialization phase is introduced optionally in ROI-NSGA-II. This phase can be skipped if a designer has good knowledge in advance about identifying points within the ROI in the early optimization phase. However, all the experiments performed in this thesis use all three phases of ROI-NSGA-II to show the generality and scalability of our proposed ROI-NSGA-II irrespective of the number of objectives, size, and position of the ROI.

4.2.2 Phase II: Invitation

The invitation phase also starts with a non-dominated sorting of points using their fitness values similar to NSGA-II. After the sorting, Pareto ranks are assigned, and individuals from initial non-domination levels with higher Pareto ranks are selected directly as in NSGA-II (cp. Section 3.6.3.3). However, to select points from final relevant non-domination level from which not all individuals are allowed to survive, the crowding distance based selection is replaced with a modified Euclidean distance from a dynamic reference point z^*. The dynamic reference point is determined as a mean of maximum and minimum fitness values of the ROI points in the final relevant non-domination level f^{ROI} in each objective, as shown in Figure 4.3. Therefore, the dynamic reference point z^* can be formulated as

$$z_i^* = \frac{max(f_i^{ROI}) + min(f_i^{ROI})}{2}, \forall i \in [1, m]. \tag{4.1}$$

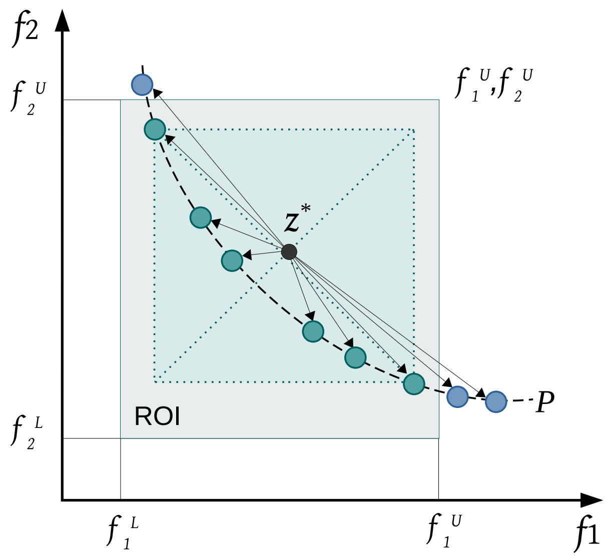


Figure 4.3: An example of dynamic reference point in ROI-NSGA-II invitation phase

Thereafter, a modified Euclidean distance between this dynamic reference point and each point in final relevant non-domination level is computed after a normalization operation. The normalization uses the difference between maximum and minimum fitness values of the ROI points f^{ROI} in each objective to weight them on a similar scale during the Euclidean distance estimation, irrespective of their actual values. For an example final relevant non-domination level P, the modified Euclidean distance d_i of an i-th point is calculated as

$$d_{i,ROI-NSGA-II} = \sqrt{\sum_{j=1}^{m} \left(\frac{f_j^i - z_j^*}{\left(max(f_i^{ROI}) - min(f_i^{ROI})\right)} \right)^2}. \tag{4.2}$$

To select a desired number of points from the final relevant non-domination level, the points are sorted based on the Euclidean distance from the dynamic reference point, and the desired number of points with minimum Euclidean distance is selected in the invitation phase. The genetic operations on these selected points close to the dynamic reference point might produce many offspring around the reference point within the ROI. Therefore, the invitation phase aggregates many points into an ROI and thereby increases the search pressure within the ROI.

4.2.3 Phase III: Conquest

The invitation phase iteratively invites many points into an ROI until the required number of points from the final relevant non-domination level can be selected entirely from the ROI. Thereafter, ROI-NSGA-II switches to a third conquest phase. Similar to the other two phases, a non-dominated sorting is performed, and points are selected directly from the initial fronts based on Pareto ranks. Since a desired number of points from the final relevant non-domination level can be selected directly from the ROI, the points outside of the ROI are discarded. Thereafter, a sorting operation is performed based on fitness values, and a crowding distance is estimated only for the points within the ROI. Finally, the required number of points is selected based on higher crowding distance values.

A crowding distance based selection ensures diversity in selected points within an ROI. Therefore, the search pressure is limited to the ROI but distributed simultaneously across the ROI. This accelerates the convergence of points identified within the ROI and conquers optimal points inside the ROI efficiently compared to a classical NSGA-II approach. During an optimization using ROI-NSGA-II, these three phases may alternate depending on the number of points identified within an ROI.

The three-phase approach in the ROI-NSGA-II inherits the properties of NSGA-II and guides the optimization efforts into an ROI. However, irrespective of other methods which inherit NSGA-II properties, our proposed ROI-NSGA-II requires only threshold values of each objective that define an ROI as input from a decision-maker. In many real-world applications, this can be provided efficiently to explore a desired ROI compared to other types of preference information. If the threshold values are not provided for any of the upper or lower, or both of these bounds for an optimization objective, the ROI-NSGA-II replaces these threshold values either with a maximum or minimum fitness value identified in a final relevant non-domination level depending on the type of threshold values. Therefore, if none of the threshold values are provided, the ROI-NSGA-II directly moves into the conquest phase and works as classical NSGA-II. These properties make our proposed ROI-NSGA-II approach scalable to different ROI sizes and adopt different types of ROIs defined using threshold values. In general, ROI-NSGA-II approach is beneficial in real-world applications such as approximate image processing where a decision-maker does not know a Pareto-optimal front in advance, whereas acceptable threshold values are known for at least some optimization objectives.

4.3 ROI-NSGA-II AxCGA

The ROI-NSGA-II AxCGA is similar to an NSGA-II AxCGA described in Chapter 3. The significant difference is that the NSGA-II selection algorithm, which is the core part of the DSE in AxCGA, is replaced with the new ROI-NSGA-II selection algorithm. Since a designer has to specify the acceptable threshold values for each objective in ROI-NSGA-II for a typical application, this information has to be additionally provided during an application-specific initialization phase. Apart from that, ROI-NSGA-II initializes and configures similar to the NSGA-II approach in the DSE experiments and does not require any additional design changes.

In approximate image processing applications, a designer is often interested in analyzing the quality-resource trade-off in a region between the quality of a reference implementation and quality-threshold parameter Q_{th}. The reference implementation quality is represented as either minimum possible error or maximum possible quality in an application, depending on chosen quality metric. For example, if quality is measured using PSNR, a reference quality value would be ∞. In contrast, the quality of a reference implementation in ΔE is zero. An optimization continuously improves towards this reference quality in approximate computing applications, and the quality of evolved points cannot be better than this value. Therefore, a designer can optionally skip specifying this second ROI bound corresponds to the minimum error or maximum quality. In such case, the ROI-NSGA-II will replace this skipped value with the minimum error or maximum quality identified in the final relevant non-domination level. This has the same effect in the algorithm as specifying reference quality as one of the quality bounds in approximate image processing applications. Therefore, the quality-threshold parameter Q_{th} is more important than the other ROI bound in the quality dimension. However, if a designer is not interested in determining the quality-resource trade-off in a region close to the reference quality, both the quality-threshold values must be specified in the ROI-NSGA-II AxCGA. Similarly, the minimum possible resource utilization in the resource objective can also be optionally skipped since the optimization can never reach a better point, and this value is also replaced with the minimum resource consumption value identified in the final relevant non-domination level.

4.4 Case Studies

To demonstrate the effectiveness of the ROI-NSGA-II AxCGA, we considered both RGB to $YCbCr$ conversion and display rendering application introduced in Chapter 3. The following sections describe the ROI setup used in these applications and discuss the obtained results from ROI-NSGA-II AxCGA. Additionally, the performance of the ROI-NSGA-II AxCGA is compared with the NSGA-II based approach and the state-of-the-art autoAx approach.

4.4.1 Case Study 1: *RGB* to *YCbCr* Color Space Conversion

We initially performed ROI-NSGA-II AxCGA on RGB to $YCbCr$ case study described in Section 3.7.1. The DFG representation of the application, approximation techniques and parameters used to generate candidate DFGs, DFG encoding, and employed genetic operations are the same as described before. Similarly, our ROI-NSGA-II based DSE experiments also used the same objective function as before, which minimizes power consumption of the design while maximizing application quality as PSNR. In the initialization and setup phase, we employed the same per-unit power consumption, toggle rate, and random samples of pixels extracted from the *Color Wheel* image as the training dataset. We used the adaptive AxCGA version with all GA-specific initialization parameters same as before. Using a similar experimental setup allows comparison of the results obtained from ROI-NSGA-II AxCGA with both the NSGA-II AxCGA and autoAx DSE.

4.4.1.1 ROI Setup

In ROI-NSGA-II AxCGA experiments, we additionally define an ROI, where a designer can select useful design points in the RGB to $YCbCr$ application, during the application-specific initialization phase. We chose the quality-threshold Q_{th} as 30 dB since a PSNR of 30 dB and above is considered acceptable in many image processing applications [14]. The points with PSNR values below this threshold are irrelevant in this case study. Due to the maximization of the quality objective in the DSE, this $Q_{th} = 30$ defines the lower threshold $f^{L}_{2=Q}$. We kept the upper threshold $f^{U}_{2=Q}$ as `None` since we are interested in a region defined between this Q_{th} and the maximum quality. As described in the previous sections, the ROI-NSGA-II handles such cases by replacing this value with the maximum quality value identified in the final relevant non-domination level in each generation. In this case study, the maximum possible quality value is ∞ in PSNR, which is approximated as 137.23 dB PSNR with a theoretical pixel difference of 0.5 in one channel for a better numerical representation. In the resource objective, the upper threshold $f^{U}_{1=R}$ is defined with the power consumption of the reference system $R(ref)$, which is equal to 26.21 mW. Since the resource objective is to minimize system power consumption, we selected $f^{L}_{1=R}$ as `None` in the ROI-NSGA-II.

4.4.1.2 DSE Results

We repeated the ROI-NSGA-II AxCGA experiments 50 times independently to demonstrate the consistency of results, and the obtained Pareto points from all these experiments are depicted in Figure 4.5a. Since the ROI-NSGA-II AxCGA is only interested in points identified within an ROI, all the points outside of this ROI are discarded from further analysis. The DSE experiments resulted in 259.96 points on average within the ROI, spanning between 211 and 301 points in individual experiments. We additionally replotted the points obtained from 50 independent experiments of NSGA-II AxCGA and autoAx DSE described in Chapter 3 also in the same figure. However, all the points outside the ROI are discarded from these results also since such points are not interesting to the designer. On average, autoAx DSE identified 243.5 points per experiment within the ROI, and NSGA-II AxCGA identified 246.78 points per experiment from 50 runs. From the figure, it can be seen that the overall trend of points obtained from the ROI-NSGA-II experiments are visually similar to the NSGA-II AxCGA experiments except in some regions. In contrast, the autoAx DSE performs poorly in a region close to the maximum application quality.

4.4.1.3 Performance Comparison between ROI-NSGA-II, NSGA-II, and autoAx

To further compare the performance of each approach, we computed hypervolume values from all independent experiments. However, we are interested only in the points evolved within the relevant ROI. Therefore, we computed the hypervolume values of these ROI points using boundary points as hypervolume reference. The hypervolume values estimated from the points within the ROI are further referred as ROI-hypervolume and the reference point defined with the ROI bounds is further denoted as ROI-hypervolume reference in the following sections. An example of such ROI-hypervolume computation is shown in Figure 4.4 in the case of a minimization problem. In this study, we used a PSNR of 30 dB, representing the lower threshold in the maximization

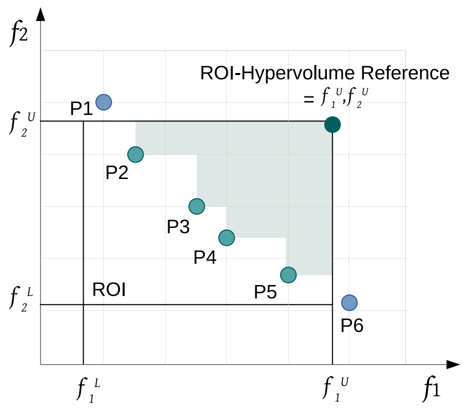


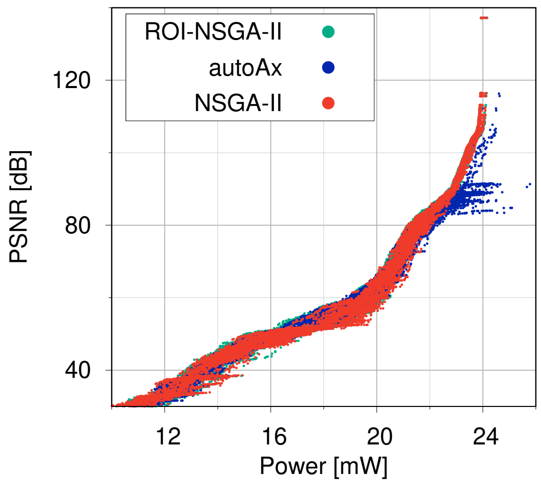
Figure 4.4: Example of the ROI-hypervolume which is represented by the shaded area for Pareto points P2 to P5 with respect to the ROI-hypervolume reference point

objective, and a power of 26.21 mW representing the upper threshold in the minimization objective as the coordinates of ROI-hypervolume reference. In general, the ROI-NSGA-II based optimization inherently tries to improve in the direction of counterpart thresholds of this reference point.

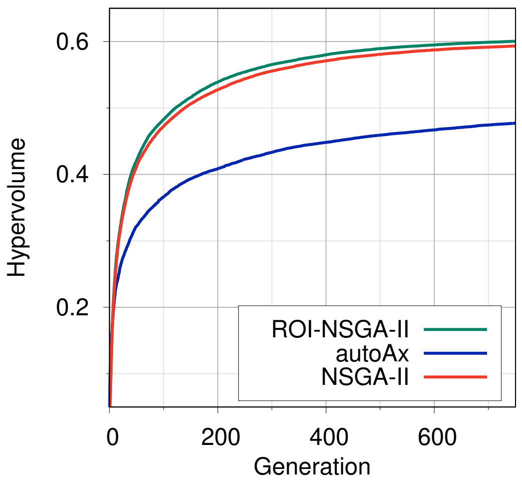
Figure 4.5b shows the average ROI-hypervolume values obtained from 50 independent experiments of ROI-NSGA-II and NSGA-II AxCGA and autoAx DSE. Since the time required for a fitness evaluation is multiple orders of magnitude higher than the execution time of the DSE algorithm, 100 iterations in autoAx DSE is considered approximately equivalent to a single AxCGA DSE generation which also includes 100 fitness evaluations. From the hypervolume plots, it can be seen that the trend of both the ROI-NSGA-II and NSGA-II based AxCGA is considerably similar with final ROI-hypervolume values 0.6004 and 0.5932. The reason for such a comparable ROI-hypervolume is due to the fact that the ROI in the RGB to $YCbCr$ conversion includes most of the global design space identified in Figure 3.15. In contrast to the performance of both AxCGA versions, the average ROI-hypervolume values obtained from autoAx DSE experiments are extremely poor, with a final ROI-hypervolume value of 0.4774 since it failed to identify many of the points near the maximum PSNR region. The average of maximum ROI-hypervolume obtained from autoAx DSE can be obtained around 100 generations in both AxCGA versions. Overall, the ROI-NSGA-II AxCGA outperforms NSGA-II based AxCGA consistently in a small margin from around 25 generations, whereas both these approaches perform significantly better than the autoAx DSE.

4.4.2 Case Study 2: Display Rendering Pipeline

We ran the ROI-NSGA-II AxCGA on the display rendering case study described in Section 3.7.2. The DSE objectives are selected as the same for ROI-NSGA-II AxCGA experiments to compare the results obtained from these experiments with NSGA-II AxCGA and autoAx DSE results. In addition, we used a similar experimental setup explained before, including approximation techniques and



(a) PSNR [dB] vs. Power.



(b) Average ROI-hypervolume over DSE generations. A single AxCGA generation is equivalent to 100 iterations in autoAx

Figure 4.5: DSE results obtained within the ROI from 50 independent experiments on RGB to $YCbCr$ conversion

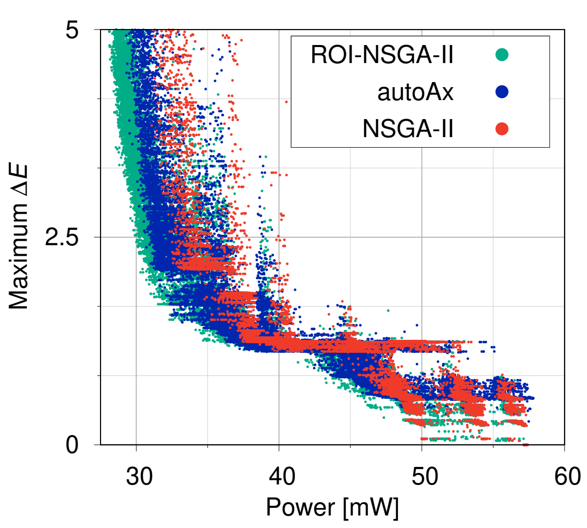
parameters, genetic encoding and operations, GA-specific initialization, area and power model parameters, and training images for the ROI-NSGA-II AxCGA experiments for a fair performance comparison. The only difference is that the integration of the threshold values of ROI-NSGA-II is required in the application-specific initialization of AxCGA, which will be described in the following section.

4.4.2.1 ROI Setup

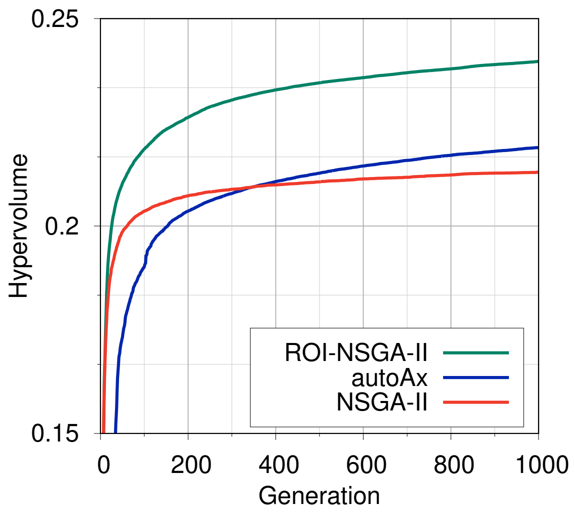
The perceivable color difference of human eyes defines the quality boundaries or the threshold values in the display rendering application. The quality in display rendering application is measured as ΔE, and the visual perception threshold of human eyes lies approximately at 2.15 ΔE [241]. Therefore, in ROI-NSGA-II AxCGA experiments, we chose ΔE of 5 as the quality-threshold in the *maximum*(ΔE) objective, which defines the worst case error from a single pixel and 2.15 ΔE as the quality-threshold in the *mean*(ΔE) objective. Since the optimization objectives are minimizing the quality error and resource consumption, these quality-thresholds define the upper threshold of the respective ROI. Any points having ΔE values above either of these quality-thresholds are not interesting for a designer in this application. Therefore, the quality-thresholds are $f^U_{2=\text{maximum}(\Delta E)}$ = 5 and $f^U_{3=\text{mean}(\Delta E)}$ = 2.15 for the ROI-NSGA-II. The ideal lower threshold of an ROI in case of applications with error metrics as quality metrics is zero since the quality error from a reference implementation is always zero. However, we kept the lower quality-thresholds $f^L_{2=\text{maximum}(\Delta E)}$ = $f^L_{3=\text{mean}(\Delta E)}$ = `None` since this is equivalent to setting up to the zero error in ROI-NSGA-II. Similar to the previous case study, the upper threshold in the power dimension is defined with the reference power consumption, and the lower threshold is kept as `None`. Therefore, we chose $f^U_{1=R}$ = 57.82 and $f^L_{1=R}$ = `None` in our experiments.

4.4.2.2 DSE Results

To analyze the results consistently, we performed ROI-NSGA-II AxCGA independently 50 times on the display rendering case study. The aggregated Pareto points from all these experiments are shown in Figure 4.6a. The evolved points within the ROI from independent experiments span between 483 and 894, with an average of 763.1 points per experiment. To further analyze and compare the ROI-NSGA-II AxCGA results, we plotted the points identified within the ROI from the 50 different experiments of NSGA-II AxCGA and autoAx DSE in the same figure. Based on the evolved points from each approach, it can be seen that the points identified using ROI-NSGA-II AxCGA are well converged and distributed throughout the ROI compared to the other two approaches. Among the NSGA-II AxCGA and autoAx DSE, the points evolved from autoAx experiments are better converged and densely populated compared to the NSGA-II AxCGA, except the region close to the maximum quality where autoAx DSE failed to identify many points. Overall, autoAx DSE identified 736.54 points per experiment within the ROI from 50 experiments, whereas only 259.8 points were identified within the ROI on average in NSGA-II AxCGA.



(a) Maximum ΔE vs. Power.



(b) Average ROI-hypervolume over DSE generations. A single AxCGA generation is equivalent to 100 iterations in autoAx

Figure 4.6: DSE results obtained within the ROI from 50 independent experiments on display rendering application

4.4.2.3 Performance Comparison between ROI-NSGA-II, NSGA-II and autoAx

The average ROI-hypervolume estimated from 50 independent experiments of ROI-NSGA-II AxCGA, NSGA-II AxCGA, and autoAx DSE are plotted in Figure 4.6b. From the figure, it can be seen that the ROI-hypervolume values over the generations from ROI-NSGA-II AxCGA significantly dominate both the NSGA-II AxCGA and autoAx DSE. The final ROI-hypervolume values are 0.2397, 0.2130, and 0.2189, respectively, in each approach. While comparing the ROI-hypervolume values of NSGA-II AxCGA and autoAx DSE, it is evident that the NSGA-II performs better than the autoAx DSE in initial generations. However, the autoAx DSE outperforms NSGA-II AxCGA from 346 generations. In the display rendering application, the ROI size is very small compared to the overall design space, and the ROI includes the knee of the quality-resource trade-off, where many possible points can exist with slightly different parameter configurations. Therefore, autoAx DSE identified many of such points and improved them using a neighborhood search method. In contrast, the search pressure is equally distributed to the entire design space in NSGA-II AxCGA, which causes less converged points and poor final ROI-hypervolume values within a small ROI. Overall, the final average ROI-hypervolume obtained from NSGA-II AxCGA and autoAx DSE can be achieved with 64 and 104 generations, respectively, of the ROI-NSGA-II AxCGA. These performance values show the scalability of ROI-NSGA-II approach to a desired ROI by maintaining both the diversity and convergence within the ROI.

4.5 Comparison between Adaptive and Non-Adaptive ROI-NSGA-II AxCGA

To demonstrate how well our adaptive GA hyperparameter approach performs on the ROI-NSGA-II experiments, we further compare the experimental results of the ROI-NSGA-II AxCGA from both the adaptive and non-adaptive versions. We use the experimental results shown in Figure 4.5 and Figure 4.6 for the adaptive ROI-NSGA-II AxCGA in RGB to $YCbCr$ conversion and display rendering case study. Additionally, we ran experiments on these case studies with the non-adaptive version of the ROI-NSGA-II AxCGA for performance comparison. The non-adaptive version uses commonly-used GA hyperparameters such as $p_m = 0.3$, $p_c = 0.7$, $\mu = 50$, and $\lambda = 100$ [232] in the experiments. The following sections discuss the results obtained from these experiments.

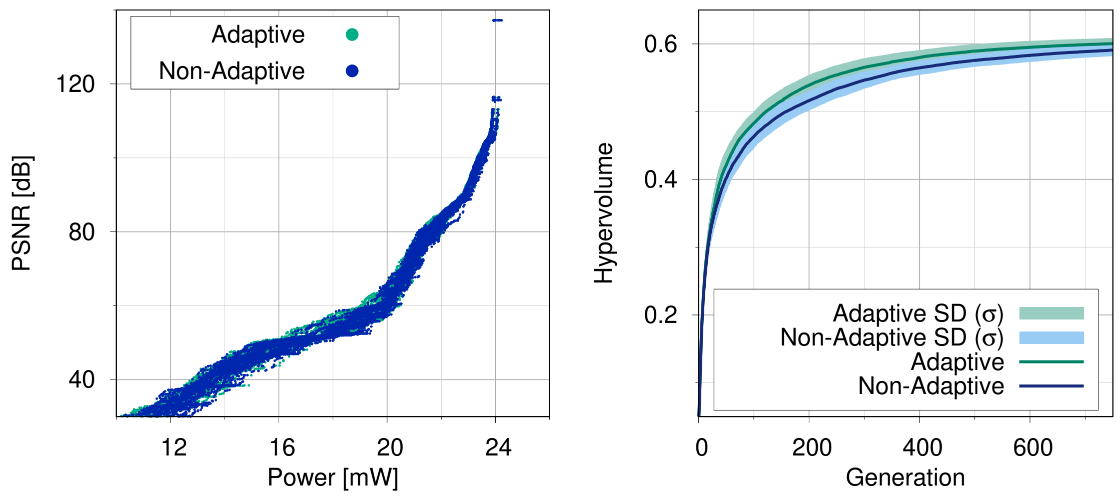
4.5.1 Case Study 1: *RGB* to *YCbCr* Color Space Conversion

Pareto points obtained from 50 independent experiments of both adaptive and non-adaptive ROI-NSGA-II AxCGA on the RGB to $YCbCr$ case study are shown in Figure 4.7a. Both approaches identified a well-distributed set of Pareto points within the ROI, and the overall trends of these points are closely comparable. However, in some regions, for example, the points identified between 16 mW and 20 mW of power and in a region close to the lower quality bound of 30dB, the adaptive approach identified better-converged points than the non-adaptive approach.

Comparing average ROI-hypervolume values between both approaches in Figure 4.7b, it can be seen that the adaptive version outperforms the non-adaptive version with final ROI-hypervolume values 0.6004 and 0.5908, respectively. However, overall trend of these curves is closely comparable with average standard deviations(σ) 0.0132 and 0.0128 in adaptive and non-adaptive approaches.

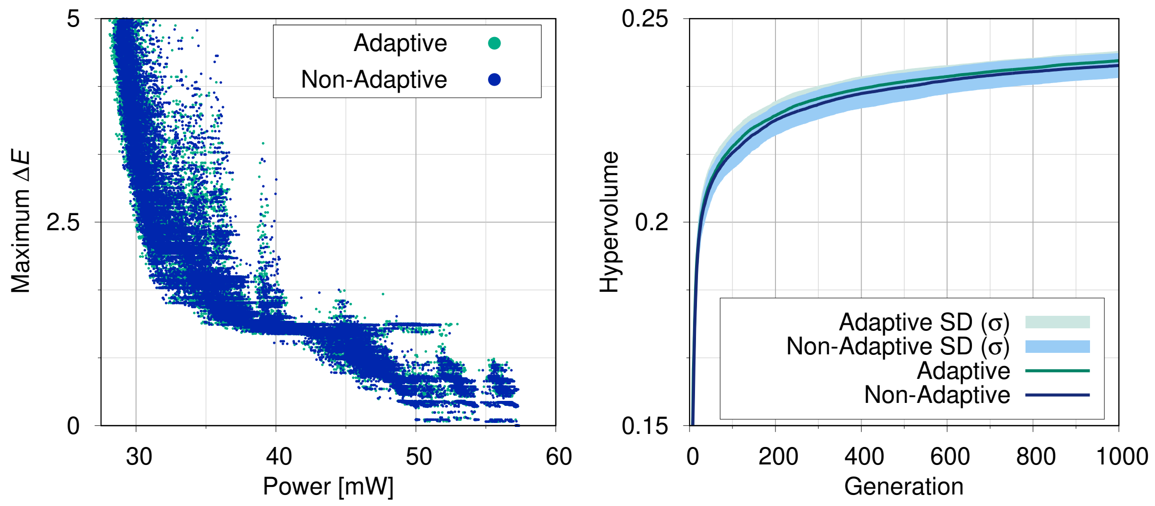
4.5.2 Case Study 2: Display Rendering Pipeline

We ran non-adaptive version of ROI-NSGA-II AxCGA on the display rendering case study as well 50 times independently to compare the results with the adaptive version. The Pareto points evolved from all experiments of both the adaptive and non-adaptive AxCGA are depicted in Figure 4.8a. From the figure, it can be seen that the distribution of points obtained from both approaches is quite similar. A comparison of average ROI-hypervolume values over generations in Figure 4.8b shows that the trend of optimization improvements in both approaches is also similar, with final ROI-hypervolume values 0.2397 and 0.2385 and average standard deviations 0.0033 and 0.0038 in adaptive and non-adaptive AxCGA. Overall, comparing adaptive and non-adaptive AxCGA results shows that the adaptive approach slightly outperforms non-adaptive AxCGA in terms of final ROI-hypervolume values, even though the general performance can be considered quite comparable.



(a) Pareto fronts obtained from 50 experiments of adaptive and non-adaptive ROI-NSGA-II AxCGA (b) Average hypervolume from 50 experiments of adaptive and non-adaptive ROI-NSGA-II AxCGA

Figure 4.7: Adaptive and non-adaptive ROI-NSGA-II AxCGA results on RGB to $YCbCr$ conversion



(a) Pareto fronts obtained from 50 experiments of adaptive and non-adaptive ROI-NSGA-II AxCGA (b) Average hypervolume from 50 experiments of adaptive and non-adaptive ROI-NSGA-II AxCGA

Figure 4.8: Adaptive and non-adaptive ROI-NSGA-II AxCGA results on display rendering pipeline

4.6 Summary

This chapter presents our three-phase ROI-NSGA-II selection method for concentrating the optimization effort into an ROI defined by a designer. The first NSGA-II initialization phase identifies initial points within the ROI and ensures a minimum guarantee of overall results that an NSGA-II can deliver. Afterward, a second invitation phase with modified Euclidean distance selection invites and aggregates many points into the ROI. Finally, a third conquest phase uses a crowding distance based selection within the ROI, which enhances the diversity and convergence of identified points within the ROI. The novel ROI-NSGA-II is integrated into our AxCGA to avoid spending DSE effort on finding irrelevant solutions in real-world approximate image processing applications. We demonstrated the usability of our ROI-NSGA-II AxCGA on both the RGB to $YCbCr$ conversion and display rendering case studies. We compared the results of ROI-NSGA-II AxCGA with NSGA-II AxCGA and state-of-the-art autoAx DSE in terms of average hypervolume values computed from the evolved points within the ROI. The hypervolume values over generations show that ROI-NSGA-II AxCGA achieved a higher final hypervolume within the ROI compared to both the other approaches. Additionally, we identified that the performance gap between ROI-NSGA-II and NSGA-II AxCGA narrows once the size of ROI increases, and this property demonstrates the scalability of ROI-NSGA-II approach for various ROI sizes. Finally, we compared the average hypervolumes obtained from both the adaptive and non-adaptive versions of ROI-NSGA-II AxCGA to analyze the performance of each approach. Comparing the average hypervolume from both approaches shows that the adaptive approach is marginally better than the non-adaptive version in both case studies.

CHAPTER 5

Verification and Validation of ROI-NSGA-II

This chapter demonstrates the functionality of novel ROI-NSGA-II approach on different optimization benchmark problems and compares the performance with a relevant state-of-the-art approach. Most of the experimental results are previously published in "Region of interest based non-dominated sorting genetic algorithm-II: an invite and conquer approach" @ACM 2022 [25]. Therefore, figures and text used in this chapter are adapted and extended from this publication.

5.1 Introduction

Conventional optimization approaches such as NSGA-II explore an entire objective space, and parameter constraints are often introduced to limit the search space and improve optimization efficiency. However, such parameter constraints cannot be provided effortlessly and are usually tied to the properties of an application. Therefore, a global optimization approach often wastes effort to find irrelevant solutions outside of a preferred region in applications where a decision-maker is interested in a region of interest. Addressing this issue, we proposed a novel ROI-NSGA-II, capable of efficiently exploring a desired part of the design space, as described in Chapter 4. The ROI-NSGA-II invites classical NSGA-II algorithm into a desired region using a modified dominance relation and conquers solutions within this region using a modified crowding distance based selection. Its potential in solving the problems is demonstrated in Chapter 4 using two approximate computing case studies. However, the performance of ROI-NSGA-II on problems with various types of Pareto-optimal fronts still needs to be investigated.

To verify and validate the performance of our ROI-NSGA-II on various problems, we consider a set of MOO benchmark problems commonly used to examine new approaches. The Pareto-optimal fronts of these problems are either continuous or disconnected, have different shapes, such as convex or concave, and have different dimensions. Verification and validation of our proposed approach in different types of problems help to demonstrate the effectiveness of our approach beyond approximate computing applications. Additionally, the ROI-NSGA-II optimization results on these problems are compared to one of the prominent state-of-the-art variants, namely R-NSGA-II [242].

The rest of this chapter is organized as follows. First, Section 5.2 gives formal definitions of different benchmark problems and briefly describes the background of the state-of-the-art R-NSGA-II approach. Thereafter, verification and validation results of ROI-NSGA-II on benchmark problems are discussed, and the optimization results are compared with R-NSGA-II in Section 5.3. This section also includes performance comparisons of both approaches on the approximate computing case studies. Finally, Section 5.4 concludes this chapter.

5.2 Background

This section formulates different MOO benchmark problems considered for performance analysis and comparison in this thesis. Furthermore, the state-of-the-art R-NSGA-II approach is described in short, and principle of Inverted Generational Distance (IGD), an additional performance indicator used in this section, is also briefly described.

5.2.1 Multi-Objective Optimization (MOO) Benchmark Problems

For verification and validation of ROI-NSGA-II, we used three different two-objective Zitzler-Deb-Thiele (ZDT) problems having distinct Pareto shapes [227], three Deb-Thiele-Laumanns-Zitzler (DTLZ)2 problems with three-to-ten objectives, and a three-objective DTLZ7 problem with sparse and disconnected Pareto front [261]. The following subsections explain each problem with its specific parameters in detail.

5.2.1.1 ZDT Test Problems

Two objective optimization benchmark problems with different Pareto-shape are common in evaluating the performance of various MOO algorithms. An ZDT test problem with two objectives can be formulated in general as,

$$\begin{aligned} \text{minimize: } & \tau(\mathrm{x}) = (f_1(x_1), f_2(\mathrm{x})), \\ \text{subject to } & f_2(\mathrm{x}) = g(x_2,\ldots,x_n)h(f_1(x_1), g(x_2,\ldots,x_n)), \\ & \mathrm{x} = \{x_1,\ldots,x_n\}. \end{aligned} \tag{5.1}$$

The function f_1 depends only on the first variable x_1 among n number of decision variables in x, and the function g depends on the remaining $n-1$ decision variables from x_2. Both the function values f_1 and g together define function value h. Therefore, different ZDT problems exist depending on these functions, number of decision variables, and the range of values these variables accept. This thesis considers three different ZDT test problems to evaluate and compare the performance of our proposed ROI-NSGA-II.

ZDT1 Test Problem

The first ZDT1 test problem $\tau_1(\mathrm{x})$ has a convex Pareto-optimal front and can be obtained from decision variables $n = 30$ using the following functions:

$$f_1(x_1) = (x_1), \tag{5.2}$$

$$g(x_2, \ldots, x_n) = 1 + 9\sum_{i=2}^{n} \frac{x_i}{(n-1)}, \tag{5.3}$$

$$h(f_1, g) = 1 - \sqrt{\frac{f_1}{g}}, \tag{5.4}$$

where x_i spans between $[0,1]$, and $g(x) = 1$ gives Pareto-optimal front for this test problem.

ZDT2 Test Problem

The ZDT2 test problem $\tau(\text{x})$ is a non-convex version of the problem with $n = 30$ decision variables, and the parameters and ranges are similar to the ZDT1 test problem, except the function $h(f_1, g)$. The $h(f_1, g)$ of ZDT2 test problem is defined as

$$h(f_1, g) = 1 - \left(\frac{f_1}{g}\right)^2. \tag{5.5}$$

ZDT3 Test Problem

The ZDT3 test problem has a discrete Pareto front consisting of several non-contiguous convex parts. Similar to other problems, the difference lies in the function h, where integration of a sinusoidal function makes it disconnected.

$$h(f_1, g) = 1 - \sqrt{\frac{f_1}{g}} - \frac{f_1}{g}\sin(10\pi f_1), \tag{5.6}$$

5.2.1.2 DTLZ Test Problems

DTLZ is a benchmark problem suite with different types of MOO problems that are scalable to a large number of objectives and decision variables. This thesis uses DTLZ2 test problems to demonstrate the performance of ROI-NSGA-II on many objective problems and a DTLZ7 test problem for demonstrating ROI-NSGA-II on a problem with three-dimensional disconnected and sparse Pareto-optimal front. Each of these test problems is explained in detail below.

DTLZ2 Test Problem

DTLZ2 problems are often used to investigate the ability to scale up the performance of a MOO algorithm with increasing number of objectives. An m objective DTLZ2 minimization problem can be formulated as

$$
\begin{aligned}
&\text{minimize: } f_1(\mathrm{x}) = (1+g(\mathrm{x}_M))\cos(x_1\tfrac{\pi}{2})\ldots\cos(x_{m-2}\tfrac{\pi}{2})\cos(x_{m-1}\tfrac{\pi}{2}),\\
&\text{minimize: } f_2(\mathrm{x}) = (1+g(\mathrm{x}_M))\cos(x_1\tfrac{\pi}{2})\ldots\cos(x_{m-2}\tfrac{\pi}{2})\sin(x_{m-1}\tfrac{\pi}{2}),\\
&\text{minimize: } f_3(\mathrm{x}) = (1+g(\mathrm{x}_M))\cos(x_1\tfrac{\pi}{2})\ldots\sin(x_{m-2}\tfrac{\pi}{2}),\\
&\qquad\vdots\\
&\text{minimize: } f_{m-1}(\mathrm{x}) = (1+g(\mathrm{x}_M))\cos(x_1\tfrac{\pi}{2})\sin(x_2\tfrac{\pi}{2}),\\
&\text{minimize: } f_m(\mathrm{x}) = (1+g(\mathrm{x}_M))\sin(x_1\tfrac{\pi}{2}),\\
&\text{where } g(\mathrm{x}_M) = \sum_{x_i\in\mathrm{x}_M}(x_i-0.5)^2, \text{ subject to } 0\le x_i\le 1, \forall\in[1,\ldots,n]
\end{aligned}
\tag{5.7}
$$

The total number of variables is given as $n = m + k - 1$, and the last k variables represent x_M in the above equation. The recommended value of k is $k = |\mathrm{x}_M| = 10$. Additionally, all individual Pareto fronts in this many-objective front are required to satisfy $\sum_{i=1}^{m} f_i = 1$, and the Pareto-optimal solutions can be obtained using $x_i = 0.5, \forall x_i \in \mathrm{x}_M$.

DTLZ7 Test Problem

DTLZ7 test problem has $2^{(m-1)}$ disconnected Pareto fronts in an m-dimensional search space. This is often used to test the capability of an algorithm to scale with complexity and maintain optimal points in disconnected regions. A DTLZ7 minimization problem can be formulated as

$$
\begin{aligned}
&\text{minimize: } f_1(\mathrm{x}_1) = x_1,\\
&\text{minimize: } f_2(\mathrm{x}_2) = x_2,\\
&\qquad\vdots\\
&\text{minimize: } f_{m-1}(\mathrm{x}_{m-1}) = x_m - 1,\\
&\text{minimize: } f_m(\mathrm{x}) = (1+g(\mathrm{x}_M))h(f_1, f_2, \ldots, f_{M-1}, g),\\
&\text{where } g(\mathrm{x}_M) = 1 + \frac{9}{|\mathrm{x}_M|}\sum_{x_i\in\mathrm{x}_M} x_i,\\
&h(f_1, f_2, \ldots, f_{M-1}, g), = m - \sum_{i=1}^{m-1}\frac{f_i}{1+g}(1+\sin(3\pi f_i)),\\
&\text{subject to } 0\le x_i\le 1, \forall\in[1,\ldots,n]
\end{aligned}
\tag{5.8}
$$

In DTLZ7, a recommended value for k is $k = |\mathrm{x}_M| = 20$, and Pareto-optimal solutions can be obtained with $\mathrm{x}_M = 0$.

5.2.2 Reference Point Based NSGA-II (R-NSGA-II)

To compare our proposed ROI-NSGA-II with a prominent state-of-the-art approach, we select R-NSGA-II, a variant of NSGA-II with modified survival selection instead of crowding distance based selection [242]. In this approach, a decision-maker specifies one or more reference points in the objective space as preference information. Initially, the R-NSGA-II also starts with a non-dominated sorting selection similar to NSGA-II. However, from the final relevant non-domination level where the NSGA-II selects individuals based on the crowding distance, the R-NSGA-II selects based on a normalized Euclidean distance from the user given reference points.

For a given reference point $\overline{z}$ in an m-objective optimization problem, the Euclidean distance of a point i is defined as

$$d_{i,R-NSGA-II} = \sqrt{\sum_{j=1}^{m} w_j \left(\frac{f_j^i - \overline{z}_j}{f_j^{max} - f_j^{min}} \right)^2}, \tag{5.9}$$

where f is the fitness value of the point i with maximum fitness f^{max} and minimum fitness f^{min} in the final relevant non-domination level, and w is a weight vector that can be used to bias some objectives more than others. During the selection of points from the final relevant non-domination level, first, points with the smallest Euclidean distance from each reference point are selected. With an additional user-given parameter epsilon ϵ, the next set of points is selected, which are at least ϵ distance apart from the initially selected points. This process continues until the required number of points are selected.

Figure 5.1 illustrates an example Euclidean distance based modified survival selection in a two-objective minimization problem. For the user-given reference points z_1 and z_2, two different groups of points are identified.

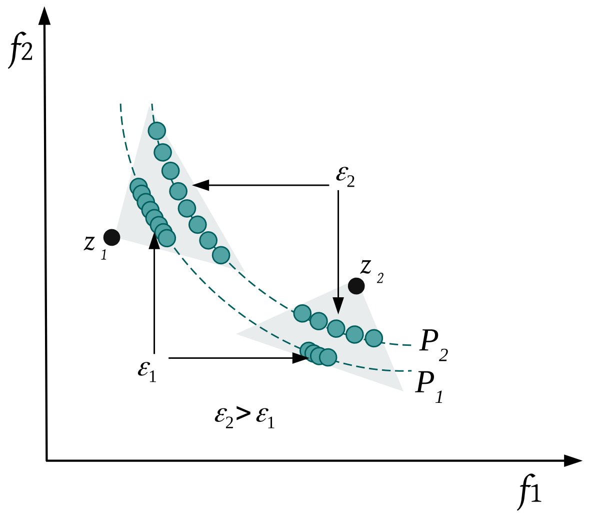


Figure 5.1: Euclidean distance based modified survival selection in R-NSGA-II

The ϵ defines spread of the selected points and indirectly influences convergence as well. A high value of ϵ leads to a broad set of points identified for each reference point, whereas a low value identifies points close to the Pareto-optimal with a lower spread. In the figure, Pareto front P_2 with a high ϵ_2 leads to identifying more diverse and less converged points. In contrast, Pareto front P_1 with a low ϵ_1 contains more converged and less diverse points. A suitable set of reference points with a careful selection of ϵ can, therefore, explore points within a specific ROI.

5.2.3 Inverted Generational Distance (IGD)

Since Pareto-optimal fronts are known in the benchmark problems, we used IGD [257] as an additional performance indicator together with hypervolume indicator to compute the performance during the experiments. The IGD accounts for an average distance from all the points in the Pareto-optimal front $O = \{o_1, \ldots, o_{|O|}\}$ to the nearest identified Pareto point $p \in P = \{p_1, \ldots, p_{|P|}\}$. Therefore, the IGD computation can be formulated as

$$\text{IGD}(P) = \frac{1}{|O|}\sqrt{\sum_{i=1}^{|O|} min(O_i - P)^2}\,. \tag{5.10}$$

Due to this average minimum distance computation, IGD values comprise both diversity and convergence measures of evolved points. Figure 5.2 shows an example of Pareto-optimal front O and nearby identified points P for each point in O.

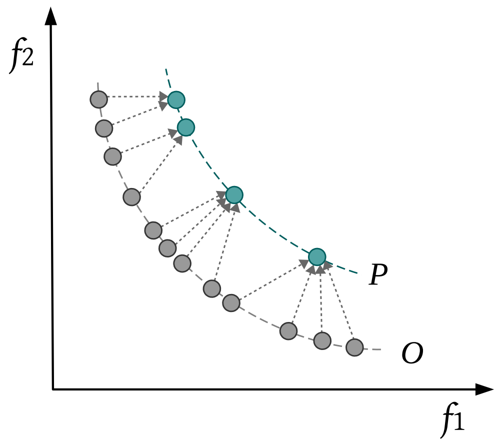


Figure 5.2: A Pareto-optimal front O and nearby identified points P in an example IGD computation

5.3 Verification and Validation Results

This section presents verification and validation results of ROI-NSGA-II on different MOO benchmark problems. We additionally compare the performance of ROI-NSGA-II with the R-NSGA-II on both the benchmark problems and approximate computing case studies. Since the principal difference between these two approaches lies in the selection technique, we used a non-adaptive version of GA with fixed hyperparameter values in all our experiments for comparison.

For benchmark problems, we employed standard genetic operations and parameter sets in our experiments. A simulated binary crossover [259] with an index of 10 and a polynomial mutation

[260] with an index of 20 are used as genetic operators. The population size μ is selected as 100, and the crossover probability p_c is set as 0.9. Due to fine-grain discrete fitness values in the benchmark problems, we used a simple evolution strategy in which mutation occurs in every generation after a crossover operation with a specified probability. However, other evolution strategies can also be applied with a similar setup for the performance comparison. The number of generations N_{gen} is selected as 500 to analyze the convergence process.

The proposed ROI-NSGA-II is compared to the R-NSGA-II based on average IGD and hypervolume from 10 independent experiments. The higher hypervolume value is better, whereas the lower IGD value is better in the performance comparison. For the ROI-NSGA-II, we initially selected an ROI, and then the parameters for the R-NSGA-II are chosen from a few trials to target the same ROI. However, more optimal choices might exist for these parameters, and determining these values often needs extensive hyperparameter searches. In fact, with our proposed ROI-NSGA-II, we overcome this hyperparameter search step by directly specifying objective thresholds to define the desired ROI in the objective space. During the performance evaluation, we discarded the evolved solutions outside of the ROI to compute both the IGD and hypervolume values. Therefore, we considered only Pareto-optimal solutions within the ROI to compute the IGD values, and the upper bound of the ROI is selected as the ROI-hypervolume reference point for the hypervolume calculations. All experiments are implemented in Python, and the benchmark problems are adopted from the pymoo framework [258]. In addition, both the Pareto front and hypervolume computations are implemented using the DEAP framework [223]. The following section describes the verification and validation results of ROI-NSGA-II on each test problem.

5.3.1 Two-Objective ZDT Problems

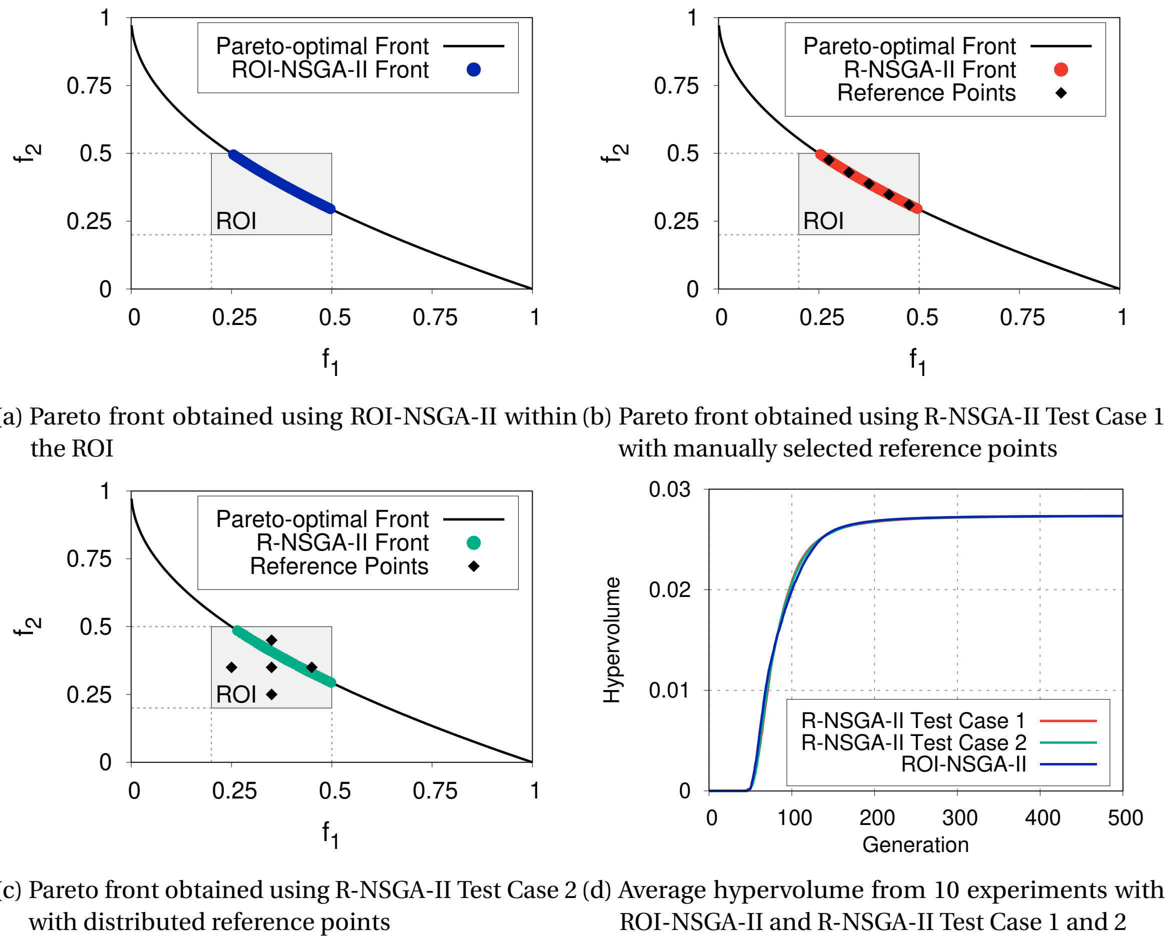
This section shows the experimental results from three different two-objective 30 variables ZDT test problems [227].

5.3.1.1 ZDT1 Test Problem

We consider ZDT1 test problem described in Section 5.2 in the first set of experiments. Since the Pareto-optimum spans continuously between $[0, 1]$ in each objective, we chose an ROI with the threshold values $f_1^L = f_2^L = 0.2$ and $f_1^U = f_2^U = 0.5$ in the experiments.

Figure 5.3a shows the points obtained from an ROI-NSGA-II experiment on the ZDT1 test problem, and it can be seen that the Pareto solutions are accurately identified within the ROI. Additionally, for the performance comparison, we used two R-NSGA-II test cases that target the same ROI using different sets of reference points. Since the Pareto-optimal front is known for this benchmark problem, we manually chose five equally spaced reference points from the Pareto-optimal front. The experiments with manually chosen reference points are further denoted as R-NSGA-II Test Case 1. Figure 5.3b depicts optimization results from an R-NSGA-II Test Case 1 experiment, and the chosen reference points are also highlighted. Similar to ROI-NSGA-II, the plot shows that the R-NSGA-II Test Case 1 could also explore a set of points within the ROI.

In the first test case, the parameters, including the reference points, are carefully chosen in a way that R-NSGA-II explores the same ROI defined for ROI-NSGA-II experiments. However,



(a) Pareto front obtained using ROI-NSGA-II within the ROI (b) Pareto front obtained using R-NSGA-II Test Case 1 with manually selected reference points

(c) Pareto front obtained using R-NSGA-II Test Case 2 with distributed reference points (d) Average hypervolume from 10 experiments with ROI-NSGA-II and R-NSGA-II Test Case 1 and 2

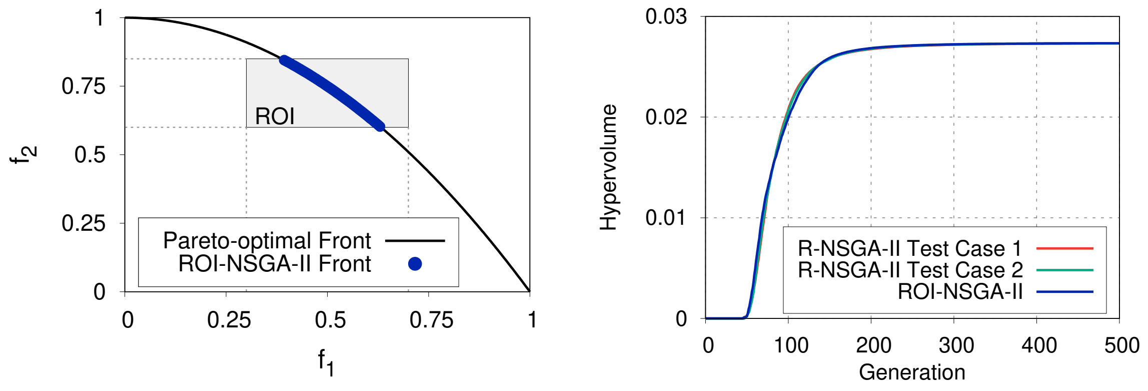
Figure 5.3: Experimental results on ZDT1 test problem with $f_1^L = f_2^L = 0.2$ and $f_1^U = f_2^U = 0.5$

a manual selection of these reference points is challenging in case of high dimensions or if a decision-maker does not have any previous knowledge about the final Pareto-optimality. To handle such a situation, we selected a strategy for an equal distribution of reference points within the ROI. Initially, the center of ROI is estimated using boundary coordinates as $f^{C_{ROI}} = f^L + \frac{f^U - f^L}{2}$. Thereafter, two additional points for each objective i are computed within the ROI as $f^{C_{ROI}} \pm \frac{f_i^U - f_i^L}{3}$. The equally distributed points used for the R-NSGA-II experiments can be seen in Figure 5.3c. Such experiments with equally distributed reference points are further denoted as R-NSGA-II Test Case 2. We ran the R-NSGA-II experiments with this setup, and the R-NSGA-II identified a similar set of Pareto points within the ROI using the distributed reference points as well. The epsilon ϵ values used for Test Case 1 and Test Case 2 are 0.007 and 0.02, respectively.

The performance comparison of each set of experiments over generations can be seen in Figure 5.3d in terms of average hypervolume values. In general, the ROI-NSGA-II shows a similar trend to R-NSGA-II with both sets of reference points in ZDT1 problems. In addition, average IGD values computed from final generation are reported in Table 5.1. From the table, it can be seen that ROI-NSGA-II has 22.6% less IGD compared to the best performed R-NSGA-II Test Case 1. However, the order of magnitude of the IGD values obtained for all three approaches are small and in a similar range. This indicates that both ROI-NSGA-II and R-NSGA-II perform well in ZDT1 problems with a slight performance improvement on ROI-NSGA-II.

5.3.1.2 ZDT2 Test Problem

Similar to ZDT1, both objectives of ZDT2 spans between $[0,1]$. Therefore, we selected an ROI using threshold values $f_1^L = 0.3$, $f_2^L = 0.6$ and $f_1^U = 0.7$, $f_2^U = 0.85$. For the performance analysis, we manually selected five equally spaced reference points from the Pareto-optimal front for Test Case 1 and used equally distributed reference points in Test Case 2. The epsilon ϵ values are chosen as 0.0068 and 0.01, respectively, for R-NSGA-II Test Case 1 and 2.



(a) Pareto front obtained for the ZDT2 problem using ROI-NSGA-II

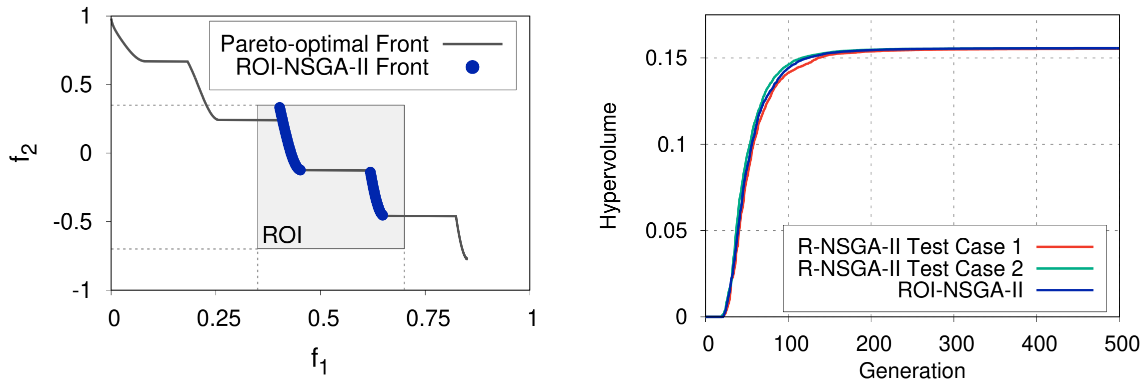
(b) Average hypervolume from 10 experiments with ROI-NSGA-II and R-NSGA-II Test Case 1 and 2

Figure 5.4: Experimental results on ZDT2 test problem with $f_1^L = 0.3$, $f_2^L = 0.6$ and $f_1^U = 0.7$, $f_2^U = 0.85$

We ran the experiments 10 times for each approach, and the Pareto front obtained from a randomly chosen ROI-NSGA-II optimization is depicted in Figure 5.4a. It can be seen that the ROI-NSGA-II is capable of identifying the Pareto-optimal points within the specified ROI in non-convex problem. We exclude the Pareto front plots from R-NSGA-II test cases since they are similar to ROI-NSGA-II in ZDT test problems. In addition, Figure 5.4b shows the average hypervolume over generation from each approach. Overall, all three approaches exhibit a similar trend with slight variations in the initial generations based on the average hypervolume values, and convergence is reached in all the approaches around 200 generations. Further analyzing average IGD values in Table 5.1, we found that the IGD from the ROI-NSGA-II is 11.7% better compared to the best performed R-NSGA-II Test Case 2.

5.3.1.3 ZDT3 Test Problem

ZDT3 test problem has a disconnected Pareto front, and one of the objectives spans between [-1, 1], whereas the second objective spans between [1, 0]. For the experiments with ROI-NSGA-II, therefore, threshold values are chosen as $f_1^L = f_2^U = 0.35$, $f_2^L = -0.7$ and $f_1^U = 0.7$ to define a relevant ROI. Similar to previous experiments, reference points are picked manually from the Pareto-optimal front as well as assigned with equal distribution within the ROI in each R-NSGA-II test case. The epsilon ϵ values chosen for Test Case 1 and Test Case 2 are 0.0062 and 0.01, respectively.



(a) Pareto front obtained for the ZDT3 problem using ROI-NSGA-II

(b) Average hypervolume from 10 experiments with ROI-NSGA-II and R-NSGA-II Test Case 1 and 2

Figure 5.5: Experimental results on ZDT3 test problem with $f_1^L = f_2^U = 0.35$, $f_2^L = -0.7$ and $f_1^U = 0.7$

Among 10 different ROI-NSGA-II experimental results, a randomly selected Pareto front is depicted in Figure 5.5a. From the figure, it is evident that the ROI-NSGA-II is capable of extracting the disconnected Pareto points within an ROI. Additionally, our proposed approach identified points which are locally optimal within the ROI and not a part of the Pareto-optimal solutions of the ZDT3 test problem. Therefore, ROI-NSGA-II can explore locally optimal points within the regions where a designer is typically interested. We exclude Pareto front plots from R-NSGA-II test cases here also due to their resemblance to the ROI-NSGA-II fronts. Similar to other ZDT experiments, the performance of all three sets of experiments has a similar trend in terms of average hypervolume values, as shown in Figure 5.5b. However, IGD from the ROI-NSGA-II is 10.4% less than the best performed R-NSGA-II Test Case 2.

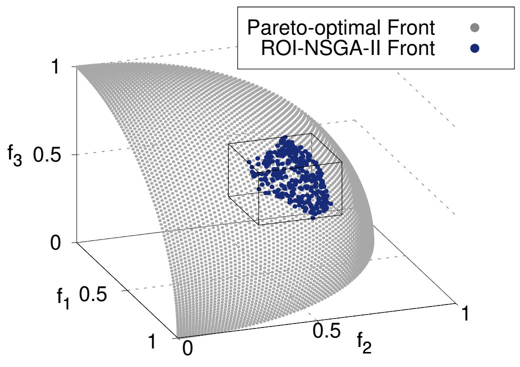
5.3.2 Many-Objective DTLZ Test Problems

This section describes the experimental setup and results of ROI-NSGA-II on different DTLZ test problems. We selected scalable DTLZ2 test problems with three, five, and ten objectives to demonstrate the capabilities of our proposed ROI-NSGA-II on many-dimensional problems. In addition, the DTLZ7 problem is also used to exemplify ROI-NSGA-II on a test problem with a three-dimensional disconnected and sparse Pareto-optimal front [261].

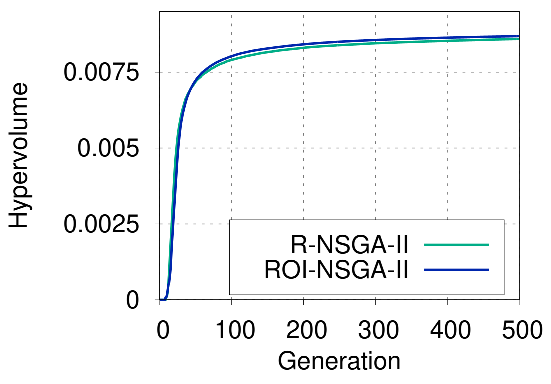
In each test problem, we initially defined a feasible ROI for the ROI-NSGA-II experiment using the upper and lower threshold values for each objective. Thereafter, we employed a set of equally distributed reference points within the ROI to explore the desired region in R-NSGA-II experiments, appending two additional reference points for each dimension. Hand-picking reference points from the Pareto-optimal front in problems with high dimensions is often difficult. In addition, the ZDT experimental results show that the average IGD values from both the R-NSGA-II test cases have a similar order of magnitude, and Test Case 2 with equally distributed reference points outperforms Test case 1 in both ZDT2 and ZDT3 test problems. Therefore, we exclude the Test Case 1 of R-NSGA-II from the DTLZ experiments. Similar to ZDT experiments, the epsilon values are also chosen to explore the whole desired region using a few preliminary trials.

5.3.2.1 Three-Objective DTLZ2 Test Problem

As a first three-objective test problem, we considered 11-variable DTLZ2 problem, which has a three-dimensional non-convex Pareto-optimal front. The threshold values which defines an ROI are selected as $(f_1^L, f_2^L, f_3^L) = 0.4$ and $(f_1^U, f_2^U, f_3^U) = 0.7$ in ROI-NSGA-II experiments. For R-NSGA-II, we used equally distributed reference points and used epsilon ϵ as 0.1 from a few preliminary trials.



(a) Pareto front obtained for the DTLZ2 problem using ROI-NSGA-II



(b) Average hypervolume from 10 experiments with ROI-NSGA-II and R-NSGA-II

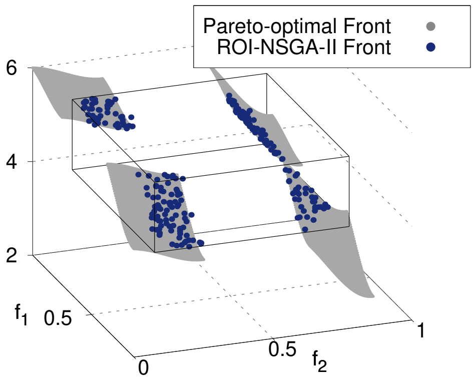
Figure 5.6: Experimental results on three-objective DTLZ2 test problem with $(f_1^L, f_2^L, f_3^L) = 0.4$ and $(f_1^U, f_2^U, f_3^U) = 0.7$

Figure 5.6a depicts a randomly selected Pareto front obtained using ROI-NSGA-II from different experiments. From the figure, it can be seen that the ROI-NSGA-II is able to find Pareto-optimal points within the ROI in the three-dimensional DTLZ2 test problem. Additionally, average hypervolume from 10 independent experiments is plotted in Figure 5.6b, and it shows that the ROI-NSGA-II has a slightly better final hypervolume than R-NSGA-II. Furthermore, Table 5.1 shows that the ROI-NSGA-II has 23.1% better average IGD compared to R-NSGA-II results. Therefore, the ROI-NSGA-II outperforms R-NSGA-II in the three-objective DTLZ2 test problem.

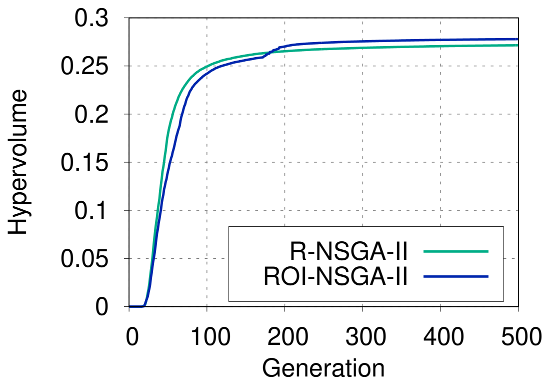
5.3.2.2 Three-Objective DTLZ7 Test Problem

To evaluate the performance of ROI-NSGA-II on a problem with a three-dimensional disconnected Pareto-optimal front, we selected an 11-variable DTLZ7 test problem. The Pareto-optimal front has four disconnected regions in the three-dimensional objective space. Therefore, we chose an ROI, which includes at least some part of each of these regions to test the performance of our

ROI-NSGA-II. The chosen threshold values for a feasible ROI are $f_1^L = 0.1$, $f_2^L = 0.1$, $f_3^L = 4$ and $f_2^U = 0.8$, $f_2^U = 0.8$, $f_3^U = 5.5$. For R-NSGA-II experiments, the epsilon ϵ is selected as 0.1.



(a) Pareto front obtained for the DTLZ7 problem using ROI-NSGA-II



(b) Average hypervolume from 10 experiments with ROI-NSGA-II and R-NSGA-II

Figure 5.7: Experimental results on three-objective DTLZ7 test problem with $f_1^L = 0.1$, $f_2^L = 0.1$, $f_3^L = 4$ and $f_2^U = 0.8$, $f_2^U = 0.8$, $f_3^U = 5.5$

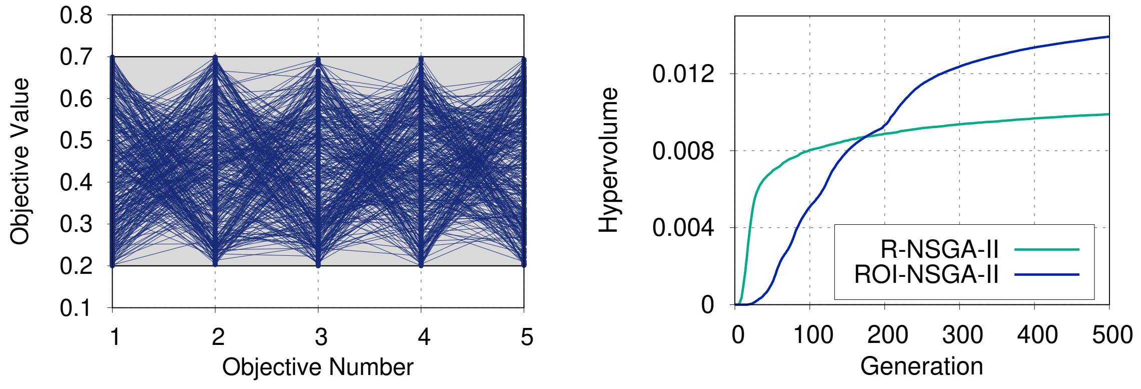
The Pareto-optimal front from a randomly selected ROI-NSGA-II experiment is shown in Figure 5.7a. From the figure, it can be seen that the ROI-NSGA-II is capable of identifying points within the ROI in the DTLZ7 problem with a disconnected three-dimensional Pareto-optimal front. In addition, hypervolume comparison between ROI-NSGA-II and R-NSGA-II in Figure 5.7b shows that the final average hypervolume in ROI-NSGA-II is higher than R-NSGA-II experiments. Similarly, average IGD values of points obtained with ROI-NSGA-II is 37.00% better than the R-NSGA-II.

Once the optimization starts, ROI-NSGA-II takes a few generations to move from the initialization phase and identify a first set of points within an ROI. As the number of objectives and problem complexity increases, identifying this first set of points within the ROI requires more generations, at least in some experimental runs. Since hypervolume values within the ROI in such runs are zero until ROI-NSGA-II moves to the invitation phase, the average hypervolume from ROI-NSGA-II experiments become lower in the initial generations. In contrast, reference points are supplied in R-NSGA-II in a way that R-NSGA-II directly explores the desired ROI. Therefore, it finds points within the ROI in the initial generations itself, which resulted in a higher average hypervolume in the initial generations. However, at a later stage, the ROI-NSGA-II outperforms R-NSGA-II with a higher average hypervolume and better final IGD values.

5.3.2.3 Five-Objective DTLZ2 Test Problems

We used 14 variables DTLZ2 test problem to demonstrate our ROI-NSGA-II on a five-objective test problem. To define a feasible ROI, we selected lower threshold values $(f_1^L, \ldots, f_5^L) = 0.2$ and upper threshold values $(f_1^U, \ldots, f_5^U) = 0.7$. The epsilon ϵ value for the R-NSGA-II is selected as 0.16. Figure

5.8a shows optimization results from one of the ROI-NSGA-II experiments, and the ROI-NSGA-II could extract diverse solutions across the ROI in DTLZ2 five-objective test problem.



(a) Pareto front obtained for the DTLZ2 problem using ROI-NSGA-II

(b) Average hypervolume from 10 experiments with ROI-NSGA-II and R-NSGA-II

Figure 5.8: Experimental results on five-objective DTLZ2 test problem with $(f_1^L,\dots,f_5^L) = 0.2$ and $(f_1^U,\dots,f_5^U) = 0.7$

Figure 5.8b shows average hypervolume values over generations from both the ROI-NSGA-II and R-NSGA-II experiments, and it can be seen that the R-NSGA-II performs better than ROI-NSGA-II until around 175 generations. As described in previous section, the ROI-NSGA-II requires more generations in high dimensional problems to identify a first set of points within an ROI, which lowers the average hypervolume in the initial generations. However, the ROI-NSGA-II hypervolume increases significantly from around 175 generations and exceeds the final hypervolume by more than 40.8% in comparison to the R-NSGA-II experiments. In addition, average IGD of ROI-NSGA-II is 52.4% lower than the R-NSGA-II results. Overall, the ROI-NSGA-II performs better than R-NSGA-II in the five-objective DTLZ2 test problem.

5.3.2.4 Ten-Objective DTLZ2 Test Problem

We analyzed the performance of our ROI-NSGA-II approach on a ten-objective DTLZ2 problem with 19 variables. An ROI is defined with lower threshold values $(f_1^L,\dots,f_{10}^L) = 0.2$ and upper threshold values $(f_1^U,\dots,f_{10}^U) = 1.2$. For R-NSGA-II experiments, we used an epsilon $\epsilon = 1.6$ from a few preliminary trials. Similar to other test problems, we repeated the experiments using both approaches 10 times each on the ten-objective DTLZ2 problem. Figure 5.9 shows the Pareto front obtained from a randomly selected ROI-NSGA-II experiment, and the ROI-NSGA-II is able to extract well-distributed solutions within the ROI. As the number of objectives increases and the number of solutions grows, the hypervolume computation becomes prohibitively long. Therefore, we could not directly compare the hypervolume values of both approaches in this test problem. However, we computed average IGD from both approaches, and the IGD of ROI-NSGA-II is 23.9% better than the R-NSGA-II experiments.

Studies on the classical NSGA-II show that a non-dominated sorting and selection approach struggles to maintain a well-distributed set of solutions in the final Pareto front in the ten-objective

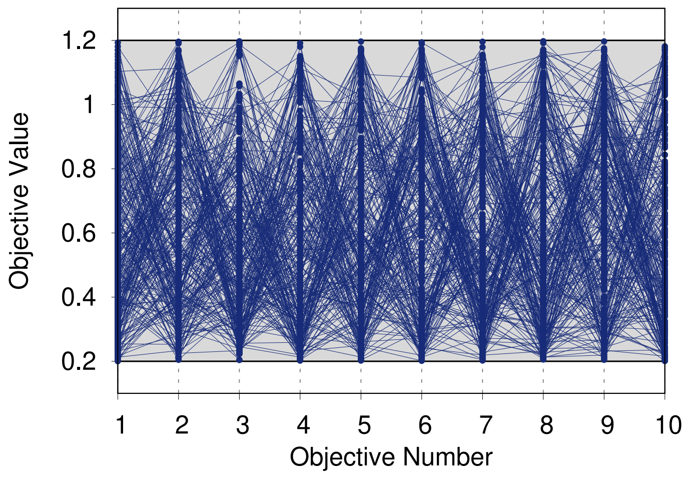


Figure 5.9: Pareto front obtained for the ten-objective DTLZ2 test problem using ROI-NSGA-II with $(f_1^L,\dots,f_{10}^L) = 0.2$ and $(f_1^U,\dots,f_{10}^U) = 1.2$

problems [262]. In contrast, our approach identifies diverse solutions by concentrating the exploration efforts into an ROI. However, we observed that the ROI-NSGA-II struggles to move into the invitation phase if the selected ROI is too small. In such cases, ROI-NSGA-II requires more generations until more than one point is found within the ROI. In the worst case, our ROI-NSGA-II approach performs equivalent to the classical NSGA-II.

Table 5.1: Average IGD values from 10 independent experiments with ROI-NSGA-II and R-NSGA-II

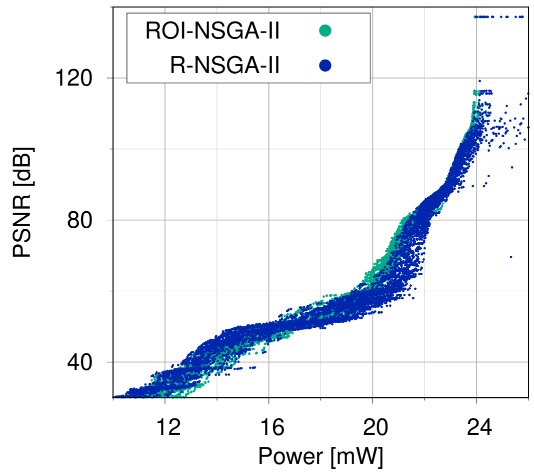
Problem	**m**	**ROI-NSGA-II**	**R-NSGA-II**	
			Test Case 1	**Test Case 2**
		ZDT Test Problems		
ZDT1	2	$\mathbf{84.24 \times 10^{-6}}$	108.87×10^{-6}	135.57×10^{-6}
ZDT2	2	$\mathbf{96.05 \times 10^{-6}}$	120.84×10^{-6}	108.72×10^{-6}
ZDT3	2	$\mathbf{174.29 \times 10^{-6}}$	224.61×10^{-6}	194.56×10^{-6}
		DTLZ Test Problems		
DTLZ2	3	$\mathbf{3.06 \times 10^{-3}}$	-	3.98×10^{-3}
DTLZ7	3	$\mathbf{30.22 \times 10^{-3}}$	-	47.96×10^{-3}
DTLZ2	5	$\mathbf{50.35 \times 10^{-3}}$	-	105.88×10^{-3}
DTLZ2	10	$\mathbf{538.96 \times 10^{-3}}$	-	708.25×10^{-3}

5.3.3 Approximate Image Processing Case Studies

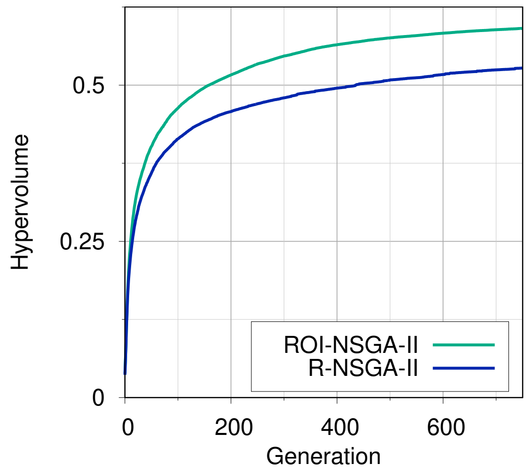
The performance of ROI-NSGA-II is already evaluated using approximate image processing case studies such as *RGB* to *YCbCr* color space conversion and display rendering application in

Chapter 4. The proposed ROI-NSGA-II AxCGA could identify solutions within the desired ROIs in both case studies more efficiently than both the autoAx DSE and NSGA-II AxCGA approaches. We additionally use R-NSGA-II selection in this chapter, which incorporates designer preference information into an optimization to compare the performance of ROI-NSGA-II on benchmark problems. Therefore, this section briefly discuss the performance comparison between ROI-NSGA-II and R-NSGA-II AxCGA on approximate image processing case studies.

We considered the same case studies and DSE setup as described in Section 3.7 for the performance comparison. However, we selected non-adaptive AxCGA versions to compare ROI-NSGA-II and R-NSGA-II approach with $\mu = 50$ and $\lambda = 100$. The crossover and mutation probabilities are set to commonly-used values of 0.7 and 0.3. Similar to previous AxCGA experiments, we chose to terminate the DSE when at least 750 and 1000 generations have been performed in color space conversion and display rendering case study. Since a designer does not know the Pareto-optimal front in such real-world applications, suitable parameters for the R-NSGA-II to explore a specific ROI cannot be supplied realistically. Hence, we used equally distributed reference points within the ROI in the R-NSGA-II experiments. We selected epsilon $\epsilon = .0.05$ and .0.01 for the color space conversion and display rendering case study based on the preliminary results from a few trials with different values. To ensure consistency of the results in these real-world case studies, we repeated the experiments 50 times for both ROI-NSGA-II and R-NSGA-II AxCGA. To compare the performance of both approaches, we computed average hypervolume over generations from all the experiments using the evolved points within the ROI by keeping the upper bounds as ROI-hypervolume reference. The IGD values cannot be computed in these problems due to the unavailability of Pareto-optimal front.



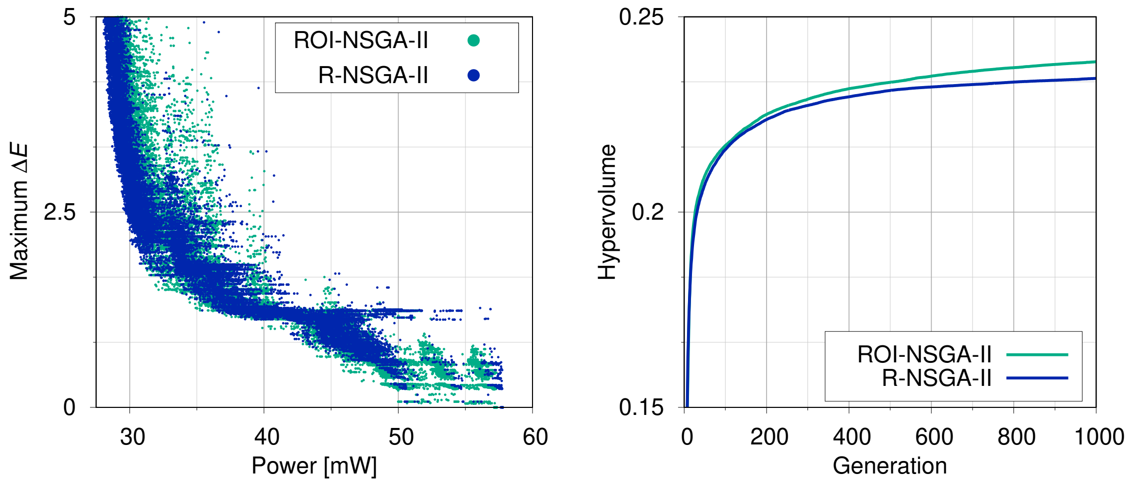
(a) Pareto fronts obtained from 50 experiments of ROI-NSGA-II and R-NSGA-II



(b) Average hypervolume within the ROI from 50 experiments of ROI-NSGA-II and R-NSGA-II

Figure 5.10: DSE results on RGB to $YCbCr$ conversion case study

Figure 5.10a shows the Pareto solutions obtained within the ROI using ROI-NSGA-II and R-NSGA-II AxCGA from 50 independent experiments on color space conversion case study. From the plot, it can be seen that both approaches are capable enough to identify the Pareto points within the ROI.



(a) Pareto fronts obtained from 50 experiments of ROI-NSGA-II and R-NSGA-II

(b) Average hypervolume within the ROI from 50 experiments of ROI-NSGA-II and R-NSGA-II

Figure 5.11: DSE results on display rendering case study

However, average hypervolume values from 50 independent DSE runs in Figure 5.10b show that the ROI-NSGA-II AxCGA clearly dominate R-NSGA-II AxCGA.

In the display rendering case study, we plotted the two-dimensional projection of solutions obtained within the ROI from 50 independent DSE experiments in Figure 5.11a, which shows the most relevant trade-off between power consumption and worst-case maximum ΔE. Even though both approaches are capable of identifying solutions within the ROI, the average hypervolume values of ROI-NSGA-II AxCGA in Figure 5.11b outperform R-NSGA-II AxCGA.

5.4 Summary

This chapter verifies and validates the performance of our proposed ROI-NSGA-II on various test problems. We considered MOO benchmark problems with Pareto-optimal fronts that are continuous or disconnected and have convex or concave shapes with different dimensions. We compared the ROI-NSGA-II optimization results on these benchmark problems with a prominent state-of-the-art R-NSGA-II approach. Since Pareto-optimal fronts are known in these problems, we computed both IGD and hypervolume values for the performance analysis. Additionally, we performed comparisons of both approaches on real-world approximate image processing case studies. In contrast, only hypervolume values are used in real-world case studies due to the unavailability of the Pareto-optimal fronts to compute IGD values. We additionally fine-tune the parameters of R-NSGA-II with preliminary experiments to explore the same desired region for a fair performance comparison. However, the experiment results show that the ROI-NSGA-II is able to explore the desired region efficiently. The overall performance of the ROI-NSGA-II is better than the fine-tuned R-NSGA-II approach in many objective benchmark problems and real-world approximate computing problems, whereas the performance is closely comparable in two- and three-objective benchmark problems.

CHAPTER 6

Conclusion & Future Work

Growing demands for hardware resources from modern-day applications inspire researchers to reconsider traditional computing methods and open up the possibilities of approximate computing that leverages application quality for resource benefits. Exploiting error resilience of applications, this thesis proposes a framework called AxCGA that can combine multiple approximations on FPGA-based image processing applications and efficiently performs DSE to determine the resource-quality trade-off. This chapter briefly outlines the overall work performed in this thesis, including the surveys presented on the state-of-the-art approximate computing methods and DSE flows, our AxCGA framework, novel ROI-NSGA-II selection algorithm which improves the AxCGA exploration efficiency, and the findings from our experiments and comparisons on two real-world applications. Finally, the current limitations of our proposals are discussed, and future works to overcome these limitations are also presented in the following sections.

6.1 Thesis Conclusion

Approximate computing is an efficient system design paradigm that overcomes the bottleneck introduced by the increased resource demands of hardware systems in data-intensive applications. Many approximate computing methods exist in the literature that address the resource demands of such applications and can design systems with reduced area, lower power consumption, and increased performance. The second chapter in this thesis includes a survey of different state-of-the-art single-purpose approximate computing methods demonstrated well in isolation and classifies distinct methods into four categories based on the targeted abstraction level, such as algorithm or software level, memory level, circuit level, and device or transistor level. Such approximated systems exploit the error resiliency of applications and trade-off the application quality for the resource benefits. Therefore, these approximate computing methods are often parameterizable to tune the strength of approximations and trade in application quality in a controllable manner.

Combining multiple approximations at different abstraction levels in an application can further increase the overall design benefits. However, this stirs up two non-trivial problems, such as the need for efficiently combining multiple parametrizable approximation methods in an application and identifying the optimal parameter configuration that delivers the best resource efficiency compared to quality degradation in an application. Addressing these problems, many design flows have been proposed in the literature that can combine multiple approximation methods and optimize joint parameterization to determine the optimal parameter configuration. The second chapter also outlines such general-purpose design flows and classifies them based on the employed approximation strategies, targeted abstraction levels, used search heuristics and search types to determine the optimal parameter configurations, and the target hardware platform. The relevance of each of these approaches has already been demonstrated in different applications, and most of these techniques can determine the quality-resource trade-off, allowing a designer to make suitable design decisions. However, none of these methods simultaneously considers many significant features such as approximations in multiple abstraction levels at the same time, efficiency and effectiveness of the optimization approach that determines the quality-resource trade-off, and architecture of FPGA. This often restricts the state-of-the-art design flows from maximizing the exploitation of quality-resource trade-off in approximate image processing applications on FPGA.

In the third chapter, we introduce AxCGA framework that can efficiently combine multiple approximations on different abstraction levels and optimally configure parameters for FPGA-based approximate image processing systems. The proposed AxCGA uses a DFG-based approach to combine multiple approximations methods in an application, where each node represents a specific operation. In AxCGA, we replace such accurate nodes with relevant approximation techniques from a pre-characterized library of approximate components and apply additional methods such as precision scaling on an application level. However, this leads to the propagation of approximation errors between the application nodes. We address this problem by globally optimizing the joint parametrization from all the approximation nodes, which inherently consider the propagation of error between the system components. This global approach indeed results in an exponential increase of design space size with overall approximation parameters, and probing all the parameter combinations to determine the resource-quality trade-off is prohibitively long. To explore such complex design spaces efficiently with a fewer number of fitness evaluations and determine the Pareto trade-off between the quality loss and resource benefits, we used a GA-based metaheuristic and NSGA-II selection in AxCGA. The genetic encoding and operations proposed in AxCGA consider the parameter interdependence to avoid the irrelevant fitness evaluations, and the modular approach enhances the generality and reusability of our approach. Additionally, we employ fast, simple, yet accurate models to accelerate the fitness evaluations during DSE. To avoid time-consuming hyperparameter optimization in our GA-based approach, we introduced a novel approach that adaptively supplies hyperparameters during the DSE.

This thesis considers a single-stage RGB to $YCbCr$ conversion and a three-stage display rendering application, where the DFG nodes of the second stage are similar to the RGB to $YCbCr$ conversion to demonstrate the applicability, modularity, and generality of our AxCGA. The first RGB to $YCbCr$ conversion case study targets to maximize the application quality as PSNR while minimizing the power consumption, whereas the second display rendering case study minimizes the quality error as mean ΔE and maximum ΔE while minimizing the power consumption. For both case studies, the execution of each AxCGA phase and the setup used in the DSE experiments are described in detail. The experimental results show that the AxCGA determine a well-distributed Pareto trade-off between the resource benefits and application quality in each experiment. A set of example solutions that satisfy an application-specific quality-threshold are synthesized for the post-synthesis power estimation and tested for quality. It can be seen that the potential power savings from these examples lie between 24.15% to 59.30% and 14.42% and 51.17%, respectively, in the first and second case studies. Additional comparison between our novel adaptive GA approach and non-adaptive approach shows a similar trend with slightly better average hypervolume values in adaptive GA over the DSE generations. Finally, performance comparisons between AxCGA and state-of-the-art autoAx based on an equivalent number of fitness evaluations show that our AxCGA outperforms the autoAx DSE. The final average hypervolume values achieved by autoAx DSE in 750 and 1000 generations can be attained around 50 and 240 AxCGA generations, respectively, in the first and second case studies.

Further analysis of the evolved Pareto points shows that the regions where a designer can select the points for practical implementation vary in applications. For example, the useful region in an RGB to $YCbCr$ conversion includes most of the design space, whereas, in the display rendering application, it is small compared to the overall design space. Therefore, a global AxCGA approach wastes the computational efforts to equally optimize the region outside of the useful region, especially in case of applications such as display rendering. To overcome this issue and concentrate the optimization effort into an ROI defined by a designer, we propose a novel three-phase ROI-NSGA-II selection method in the fourth chapter in which a designer has to specify ROI bounds additionally. The first initialization phase of the ROI-NSGA-II starts with the classical NSGA-II selection and identifies the first set of points. Thereafter, the ROI-NSGA-II moves to the invitation phase, which aims to aggregate many points into the desired ROI without any diversity mechanism. In the final conquest phase, the ROI-NSGA-II selection enhances the diversity and convergence of identified points within the ROI. In approximate computing applications, bounds of ROI can be specified directly from the quality-threshold and resource utilization of the reference implementation. The other optional bounds of ROI are determined from maximum quality or minimum quality error in the quality dimension and minimum possible resource utilization in the resource dimension. We integrated the ROI-NSGA-II into the AxCGA, repeated the experiments on both case studies, and evaluated the performance in terms of the average hypervolume of evolved points within ROI. Overall, the hypervolume values show that the ROI-NSGA-II AxCGA outperforms both the NSGA-II AxCGA and autoAx DSE in both case studies. However, the performance gap between the NSGA-II and ROI-NSGA-II AxCGA narrows when the ROI size increases, which demonstrates the scalability of our proposed ROI-NSGA-II approach. Using ROI-NSGA-II AxCGA, we also performed a comparison of adaptive and non-adaptive approaches, and the results show that the adaptive approach marginally outperforms the non-adaptive approach.

We finally tested ROI-NSGA-II on various optimization benchmark problems in Chapter 5 to verify and validate our proposed approach. The benchmark set comprises two-objective ZDT test problems with continuous or disconnected and concave- or convex-shaped Pareto-optimal fronts and three- to ten-objective DTLZ problems. We compared the performance of the ROI-NSGA-II with a relevant R-NSGA-II approach that can optimize an ROI after a time-consuming parameter tuning. The experimental results show that the ROI-NSGA-II performs equally or marginally better in two- and three-objective problems and significantly better in five- and ten-objective problems and approximate image processing case studies compared to R-NSGA-II. Overall, the ROI-NSGA-II improves AxCGA efficiency, and it can be applicable to generic optimization problems as well.

6.2 Future Works

The proposed AxCGA framework for approximate image processing on FPGA is demonstrated well in real-world image processing case studies, and experimental results outline the potential benefits of using our methodology. Integration of novel ROI-NSGA-II into the AxCGA makes our DSE approach scalable to the desired regions, and the adaptive hyperparameter approach bypasses time-consuming hyperparameter optimizations. However, many features can be appended to our framework, further enhancing the applicability to different applications and supporting additional approximation objectives. This section briefly describes the current limitations of our AxCGA and the possibilities to overcome these in the future.

The current version of our AxCGA framework is restricted to pixel or data streaming applications due to the simplicity of our resource models. We initially employed a divide and conquer approach to estimate total FPGA resources and then computed the power consumption directly from these resources. Therefore, our models cannot estimate power values accurately in an application with feedback loops among system components. To overcome this restriction, a sophisticated power model which could predict the power values from such applications can extend our AxCGA to a wide range of image processing applications, including spatial image processing such as filters or morphological operations. This extension could support many other signal-processing applications also which often include feedback loops or retrospective communications.

Since the existing library of approximate components does not include system components such as approximate dividers or comparators, the trade-off that AxCGA can exploit is limited in applications with such components. Therefore, the component library can be expanded with additional components from the literature to exploit the benefits of approximations on a diverse set of applications. Similarly, with an extension of approximate components in the library and resource models for other computing platforms, such as ASIC or GPU, the AxCGA can be easily extended to these platforms also since the DSE part is independent of the target platform.

A further significant extension of AxCGA is the integration of the delay or latency as an approximate computing objective. The current version of AxCGA primarily aims to design energy-efficient or area-efficient designs which satisfy the targeted timing constraints. However, many state-of-the-art approximate computing DSE approaches support timing models to include the delay or latency as a DSE objective. Therefore, integrating such models that can identify and estimate delay from the critical path can further extend the suitability of our AxCGA to additional design requirements.

Many state-of-the-art approaches are integrated into the standard HLS workflows to generate a DFG from a high-level description of an application. To further automate and reduce designer interventions in the AxCGA framework, a similar approach can be adopted to extend automatic DFG formation. Additionally, the component selection in AxCGA can also be automated with some additional selection algorithm similar to the state-of-the-art approximate computing approaches such as autoAx. Finally, the novel ROI-NSGA-II and adaptive GA approach need to be evaluated on additional benchmark problems and compared with other promising approaches in the future.

List of Figures

List of Tables

List of Abbreviations

ABACUS	Automated Behavioral Synthesis of Approximate Computing Systems. 33, 44
ACA	Accuracy-Configurable Approximate. 24
ACMA	Accuracy Configurable Multiplier. 26
ADC	Approximation Don't Care. 32
ALMs	approximate logarithmic multipliers. 27
AMA	approximate mirror adder. 23
A-NPU	analoge neural processing unit. 18
ANN	artificial neural network. 18, 20, 37
AppAxO	application-specific approximate operators. 25
ASIC	application specific integrated circuit. 4, 12, 22, 29, 30, 31, 36, 37, 38, 39, 40, 41, 44, 45, 93, 138
ASLAN	Automatic Methodology for Sequential Logic ApproximatioN. 33
AST	abstract synthesis tree. 33
AWG	Alexa Wide Gamut. 81, 82
AXA	approximate XOR/XNOR-based adder. 23
AxCGA	A DSE Framework for Approximate Computing Using Genetic Algorithm. 5, 6, 7, 8, 10, 29, 37, 40, 41, 45, 46, 47, 48, 49, 50, 51, 54, 55, 56, 57, 58, 59, 60, 61, 62, 65, 66, 67, 68, 69, 70, 71, 73, 75, 76, 77, 78, 79, 80, 81, 82, 84, 85, 86, 87, 88, 91, 92, 93, 94, 95, 96, 97, 99, 100, 101, 103, 105, 106, 109, 110, 111, 112, 113, 114, 115, 116, 117, 133, 134, 135, 136, 137, 138, 139, 140
AXDCnr	approximate unsigned nonrestoring array divider cell. 28
AXDnr	approximate unsigned nonrestoring divider. 28
AXHD	approximate hybrid divider. 28
BAM	Broken-Array Multiplier. 26, 51, 73, 82, 139
BBM	broken-Booth multiplier. 26

BIC	Bio-inspired Imprecise Computational block. 26
BRAM	block RAM. 19, 21, 40, 45, 46, 52, 82
CADE	Configurable Approximate Divider. 28
CCA	consistent carry approximate adder. 25
CGP	cartesian genetic programming. 23, 25, 28, 34, 41, 44
CPL	complementary pass transistor logic. 23
CPU	central processing unit. 4, 12, 41
CSPA	high-performance low-power carry speculative adder. 25
DCT	discrete cosine transform. 17
DFG	data flow graph. 5, 36, 37, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 58, 59, 60, 61, 70, 71, 72, 74, 77, 81, 82, 87, 101, 110, 136, 138, 139
DFT	discrete Fourier transform. 17
DRAM	dynamic random-access memory. 19, 20, 21, 40
DRUM	Dynamic Range Unbiased Multiplier. 26, 51
DSE	design space exploration. x, 5, 6, 7, 8, 10, 32, 35, 36, 37, 39, 41, 43, 44, 45, 46, 47, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 61, 62, 65, 66, 67, 68, 69, 76, 77, 78, 79, 80, 87, 88, 89, 91, 93, 94, 96, 97, 98, 99, 100, 101, 103, 104, 105, 106, 109, 110, 111, 112, 113, 114, 115, 117, 133, 134, 135, 136, 137, 138, 139, 140, 141
DSP	digital signal processing. 25, 45, 46, 52, 73
DTLZ	Deb-Thiele-Laumanns-Zitzler. 120, 121, 122, 128, 129, 130, 131, 132, 137, 141
EOTF	electro-optical transfer function. 80, 82, 84, 85, 86, 98
EPE	Early Power Estimator. 54
ETA	Error-Tolerant Adder. 23, 24
ETM	Error Tolerant Multiplier. 26
FA	Full Adder. 26
FaNZeD	approximate floating-point divider. 28
FFT	fast Fourier transform. 17
FPGA	field-programmable gate array. 4, 5, 6, 7, 8, 10, 12, 14, 15, 19, 20, 21, 22, 23, 25, 26, 27, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 43, 44, 45, 46, 48, 49, 50, 51, 52, 53, 54, 70, 73, 76, 79, 80, 81, 82, 83, 87, 89, 93, 101, 103, 135, 136, 138

GA	genetic algorithm. 5, 7, 8, 35, 43, 45, 46, 49, 50, 56, 57, 58, 59, 60, 61, 62, 66, 67, 69, 77, 78, 86, 87, 90, 91, 92, 101, 103, 106, 107, 110, 113, 115, 124, 136, 137, 138, 139
GDA	Gracefully-Degrading Adder. 24
GeAr	Generic Accuracy Configurable. 24
GPU	graphics processing unit. 12, 17, 22, 41, 138
HBL	horizontal-broken level. 26, 73, 74
HDL	hardware description language. 50, 52
HLS	high-level synthesis. 29, 35, 36, 39, 40, 44, 138
HOAANED	hardware optimization and a near-normal error distribution. 24, 51
IGD	Inverted Generational Distance. 120, 124, 125, 126, 127, 128, 129, 130, 131, 133, 134, 141
ILM	improved logarithmic multiplier. 27
IMPACT	imprecise adders for low-power approximate computing. 23
InXA	inexact adder. 23
INZeD	approximate integer divider. 28
K-Map	Karnaugh Map. 25
LeAp	Leading-one Detection-based Softcore Approximate Multipliers. 27
LM	Logarithmic Multiplier. 27, 51
LOA	lower-OR adder. 24, 51
LSA	lower-select adder. 24, 51, 74, 82
LSB	least significant bit. 21, 23, 26, 33, 74
LUT	look-up table. 24, 25, 27, 36, 38, 40, 45, 46, 52, 73
MA	mirror adder. 23, 74, 82
MBM	minimally biased multiplier. 27
MOO	multi-objective optimization. 45, 47, 56, 57, 103, 119, 120, 121, 124, 134
MSB	most significant bit. 21, 23, 26, 74, 87
MSE	mean squared error. 54, 56
NN	neural network. 21
NPU	neural processing unit. 18
NSGA	Non-dominated Sorting Genetic Algorithm. 104
NSGA-II	Non-dominated Sorting Genetic Algorithm-II. 5, 6, 7, 10, 34, 39, 44, 45, 50, 57, 58, 62, 63, 69, 96, 101, 103, 104, 105, 106, 107, 108, 109, 110, 111, 112, 114, 115, 117, 119, 123, 131, 132, 133, 136, 137, 139

ODC	Observability Don't Care. 32
OLOCA	optimized lower part constant-OR adder. 24, 51
PP	partial product. 25, 26, 27, 73
PSNR	peak signal-to-noise ratio. 4, 54, 56, 76, 77, 78, 79, 80, 92, 95, 96, 110, 111, 112, 136, 139
PSO	particle swarm optimization. 57
PWL	piecewise linear. 16
PWP	piecewise polynomial. 16
QCC	Quality Constraint Circuit. 32, 33
QEC	Quality Evaluation Circuit. 33
QuAd	quality-area optimal Low-Latency approximate adder. 24
QUAES	Quality Assurance, Approximation, Estimation, and Search Space Exploration. 38
RD-NSGA-II	Reference Direction Based NSGA-II. 104
R-NSGA-II	Reference Point-Based Non-dominated Sorting Genetic Algorithm-II. 104, 119, 120, 123, 124, 125, 126, 127, 128, 129, 130, 131, 133, 134, 137, 138, 140
RBA	Energy efficient Rounding-Based Approximate Multiplier. 27, 51
REALM	reduced-error approximate logarithmic multiplier. 27
RoBA	Rounding-Based Approximate Multiplier. 27, 51
ROI	region of interest. 7, 103, 105, 106, 107, 108, 109, 110, 111, 112, 113, 114, 115, 117, 124, 125, 126, 127, 128, 129, 130, 131, 132, 133, 134, 137, 140
ROI-NSGA-II	Region of Interest Non-dominated Sorting Genetic Algorithm-II. 5, 6, 7, 8, 10, 41, 103, 105, 106, 107, 108, 109, 110, 111, 112, 113, 114, 115, 116, 117, 119, 120, 121, 123, 124, 125, 126, 127, 128, 129, 130, 131, 132, 133, 134, 135, 137, 138, 140
RTL	register-transfer level. 32, 36, 53
SA	simulated annealing. 57
SALSA	Systematic methodology for Automatic Logic Synthesis of Approximate circuits. 32, 33
SASIMI	Substitute-And-SIMplIfy. 32, 33
SCALS	Statistically Certified Approximate Logic Synthesis. 36
SCSA	speculative carry select addition. 25
SEERAD	High speed yet Energy Efficient Rounding-based Approximate divider. 28
SoC	System-on-Chip. 25

SRAM	static random access memory. 19, 21
SSIM	structural similarity measure. 45, 54
SSM	Static Segment Multiplier. 26
STFT	short-time Fourier transform. 17
TG	transmission gate. 23
TOSAM	Truncation and rounding-based scalable approximate multiplier. 26
TRM	Tiered Reliability Memory. 21
UDM	underdesigned multiplier. 25
VBL	vertical-broken level. 26, 73, 74
XPE	Xilinx Power Estimator. 54
ZDT	Zitzler-Deb-Thiele. 120, 121, 125, 126, 127, 128, 129, 132, 137, 141

Bibliography

[1] G. E. Moore. Cramming more components onto integrated circuits, Reprinted from Electronics, volume 38, number 8, April 19, 1965, pp.114 ff. *IEEE Solid-State Circuits Society Newsletter*, 11(3):33–35, September 2006. ISSN: 1098-4232. DOI: `10.1109/N-SSC.2006.4785860`.

[2] 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*, 9(5):256–268, October 1974. ISSN: 1558-173X. DOI: `10.1109/JSSC.1974.1050511`.

[3] K. Rupp. 50 years of microprocessor trend data, February 2022. URL: `https://github.com/karlrupp/microprocessor-trend-data` (visited on 04/04/2023). GitHub.

[4] M. Bohr. A 30 Year Retrospective on Dennard's MOSFET Scaling Paper. *IEEE Solid-State Circuits Society Newsletter*, 12(1):11–13, 2007. ISSN: 1098-4232. DOI: `10.1109/N-SSC.2007.4785534`.

[5] IEEE NorCAS. Approximate computing: test and reliability issues and opportunities, November 2020. URL: `https://www.youtube.com/watch?v=xjUN-8yLL68` (visited on 04/04/2023). YouTube.

[6] D. Jahier Pagliari, M. Poncino and E. Macii. Energy-Efficient Digital Processing via Approximate Computing. In *Smart Systems Integration and Simulation*, pages 55–89. February 2016. ISBN: 978-3-319-27390-7. DOI: `10.1007/978-3-319-27392-1_4`.

[7] V. K. Chippa, S. T. Chakradhar, K. Roy and A. Raghunathan. Analysis and characterization of inherent application resilience for approximate computing. In *Proceedings of the 50th Annual Design Automation Conference*, DAC '13, pages 1–9, New York, NY, USA. Association for Computing Machinery, May 2013. ISBN: 978-1-4503-2071-9. DOI: `10.1145/2463209.2488873`.

[8] S. Venkataramani, S. T. Chakradhar, K. Roy and A. Raghunathan. Approximate computing and the quest for computing efficiency. In *Proceedings of the 52nd Annual Design Automation Conference*, DAC '15, pages 1–6, New York, NY, USA. Association for Computing Machinery, June 2015. ISBN: 978-1-4503-3520-1. DOI: `10.1145/2744769.2751163`.

[9] S. Reda and M. Shafique. *Approximate Circuits*. Springer International Publishing, 1st edition, 2019. ISBN: 978-3-319-99321-8. DOI: `10.1007/978-3-319-99322-5`.

[10] A. K. Mishra, R. Barik and S. Paul. iACT: A Software-Hardware Framework for Understanding the Scope of Approximate Computing. *Workshop on Approximate Computing Across the System Stack (WACAS)*, 2014.

[11] S. Hashemi, H. Tann, F. Buttafuoco and S. Reda. Approximate Computing for Biometric Security Systems: A Case Study on Iris Scanning. In *2018 Design, Automation Test in Europe Conference Exhibition (DATE)*, pages 319–324, March 2018. DOI: `10.23919/DATE.2018.8342029`. ISSN: 1558-1101.

[12] G. Karakonstantis, N. Banerjee, K. Roy and C. Chakrabarti. Design methodology to trade off power, output quality and error resiliency: application to color interpolation filtering. In *2007 IEEE/ACM International Conference on Computer-Aided Design*, pages 199–204, November 2007. DOI: `10.1109/ICCAD.2007.4397266`. ISSN: 1558-2434.

[13] G. Rodrigues, F. Lima Kastensmidt and A. Bosio. Survey on Approximate Computing and Its Intrinsic Fault Tolerance. *Electronics*, 9(4):557, April 2020. ISSN: 2079-9292. DOI: `10.3390/electronics9040557`.

[14] S. Mittal. A Survey of Techniques for Approximate Computing. *ACM Computing Surveys*, 48(4), March 2016. ISSN: 0360-0300. DOI: `10.1145/2893356`.

[15] P. Stanley-Marbell, A. Alaghi, M. Carbin, E. Darulova, L. Dolecek, A. Gerstlauer, G. Gillani, D. Jevdjic, T. Moreau, M. Cacciotti, A. Daglis, N. E. Jerger, B. Falsafi, S. Misailovic, A. Sampson and D. Zufferey. Exploiting Errors for Efficiency: A Survey from Circuits to Applications. *ACM Computing Surveys*, 53(3):51:1–51:39, June 2020. ISSN: 0360-0300. DOI: `10.1145/3394898`.

[16] Kodak Research Labs. Kodak lossless true color image suite [dataset], 1999. Retrieved from `http://r0k.us/graphics/kodak/`, accessed April 04, 2023.

[17] Q. Xu, T. Mytkowicz and N. S. Kim. Approximate Computing: A Survey. *IEEE Design Test*, 33(1):8–22, February 2016. ISSN: 2168-2364. DOI: `10.1109/MDAT.2015.2505723`.

[18] H. B. Barua and K. C. Mondal. Approximate Computing: A Survey of Recent Trends—Bringing Greenness to Computing and Communication. *Journal of The Institution of Engineers (India): Series B*, 100(6):619–626, December 2019. ISSN: 2250-2114. DOI: `10.1007/s40031-019-00418-8`.

[19] H. Jiang, F. J. H. Santiago, H. Mo, L. Liu and J. Han. Approximate Arithmetic Circuits: A Survey, Characterization, and Recent Applications. *Proceedings of the IEEE*, 108(12):2108–2135, December 2020. ISSN: 1558-2256. DOI: `10.1109/JPROC.2020.3006451`.

[20] S. Venkataramani, K. Roy and A. Raghunathan. Substitute-and-simplify: A unified design paradigm for approximate and quality configurable circuits. In *2013 Design, Automation Test in Europe Conference Exhibition (DATE)*, pages 1367–1372, March 2013. DOI: `10.7873/DATE.2013.280`.

[21] K. Nepal, Y. Li, R. I. Bahar and S. Reda. ABACUS: A technique for automated behavioral synthesis of approximate computing circuits. In *2014 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 1–6, March 2014. DOI: `10.7873/DATE.2014.374`. ISSN: 1558-1101.

[22] ACIP - Approximate Computing for Professional Image Processing. URL: `https://www.ce.cit.tum.de/lis/forschung/` (visited on 04/04/2023).

[23] M. Manuel, A. Kreddig, S. Conrady, N. Anh Vu Doan and W. Stechele. Model-Based Design Space Exploration for Approximate Image Processing on FPGA. In *2020 IEEE Nordic Circuits and Systems Conference (NorCAS)*, pages 1–7, October 2020. DOI: `10.1109/NorCAS51424.2020.9265138`.

[24] M. Manuel, A. Kreddig, S. Conrady, N. A. V. Doan and W. Stechele. Region of Interest-Based Parameter Optimization for Approximate Image Processing on FPGAs. *International Journal of Networking and Computing*, 11(2):438–462, 2021. DOI: `10.15803/ijnc.11.2_438`.

[25] M. Manuel, B. Hien, S. Conrady, A. Kreddig, N. A. V. Doan and W. Stechele. Region of interest based non-dominated sorting genetic algorithm-II: an invite and conquer approach. In *Proceedings of the Genetic and Evolutionary Computation Conference*, GECCO '22, pages 556–564, New York, NY, USA. Association for Computing Machinery, July 2022. ISBN: 978-1-4503-9237-2. DOI: `10.1145/3512290.3528872`.

[26] S. Conrady, M. Manuel, A. Kreddig and W. Stechele. LCS-based automatic configuration of approximate computing parameters for FPGA system designs. In *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, GECCO '19, pages 1271–1279, New York, NY, USA. Association for Computing Machinery, July 2019. ISBN: 978-1-4503-6748-6. DOI: `10.1145/3319619.3326820`.

[27] N. A. Vu Doan, M. Manuel, S. Conrady, A. Kreddig and W. Stechele. Parameter Optimization of Approximate Image Processing Algorithms in FPGAs. In *2020 Eighth International Symposium on Computing and Networking Workshops (CANDARW)*, pages 74–80, November 2020. DOI: `10.1109/CANDARW51189.2020.00026`.

[28] A. Kreddig, S. Conrady, M. Manuel and W. Stechele. A Framework for Hardware-Accelerated Design Space Exploration for Approximate Computing on FPGA. In *2021 24th Euromicro Conference on Digital System Design (DSD)*, pages 1–8, September 2021. DOI: `10.1109/DSD53832.2021.00010`.

[29] S. Conrady, A. Kreddig, M. Manuel, N. A. V. Doan and W. Stechele. Model-Based Design Space Exploration for FPGA-based Image Processing Applications Employing Parameterizable Approximations. *Microprocessors and Microsystems*, 87:104386, November 2021. ISSN: 0141-9331. DOI: `10.1016/j.micpro.2021.104386`.

[30] S. Xu and B. C. Schafer. Exposing Approximate Computing Optimizations at Different Levels: From Behavioral to Gate-Level. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 25(11):3077–3088, November 2017. ISSN: 1557-9999. DOI: `10.1109/TVLSI.2017.2735299`.

[31] G. Zervakis, S. Xydis, D. Soudris and K. Pekmestzi. Multi-Level Approximate Accelerator Synthesis Under Voltage Island Constraints. *IEEE Transactions on Circuits and Systems II: Express Briefs*, 66(4):607–611, April 2019. ISSN: 1558-3791. DOI: `10.1109/TCSII.2018.2869025`.

[32] S. Sidiroglou-Douskos, S. Misailovic, H. Hoffmann and M. Rinard. Managing performance vs. accuracy trade-offs with loop perforation. In *Proceedings of the 19th ACM SIGSOFT symposium and the 13th European conference on Foundations of software engineering*, ESEC/FSE '11, pages 124–134, New York, NY, USA. Association for Computing Machinery, September 2011. ISBN: 978-1-4503-0443-6. DOI: `10.1145/2025113.2025133`.

[33] H. Hoffmann, S. Misailovic, S. Sidiroglou, A. Agarwal and M. Rinard. Using Code Perforation to Improve Performance, Reduce Energy Consumption, and Respond to Failures, September 2009. Technical Report, MIT-CSAIL-TR-2009-042.

[34] S. Misailovic, S. Sidiroglou, H. Hoffmann and M. Rinard. Quality of service profiling. In *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering - Volume 1*, ICSE '10, pages 25–34, New York, NY, USA. Association for Computing Machinery, May 2010. ISBN: 978-1-60558-719-6. DOI: `10.1145/1806799.1806808`.

[35] W. Baek and T. M. Chilimbi. Green: a framework for supporting energy-conscious programming using controlled approximation. *ACM SIGPLAN Notices*, 45(6):198–209, June 2010. ISSN: 0362-1340. DOI: `10.1145/1809028.1806620`.

[36] M. Samadi, D. A. Jamshidi, J. Lee and S. Mahlke. Paraprox: pattern-based approximation for data parallel applications. *ACM SIGARCH Computer Architecture News*, 42(1):35–50, February 2014. ISSN: 0163-5964. DOI: `10.1145/2654822.2541948`.

[37] M. Samadi, J. Lee, D. A. Jamshidi, A. Hormati and S. Mahlke. SAGE: self-tuning approximation for graphics engines. In *Proceedings of the 46th Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO-46, pages 13–24, New York, NY, USA. Association for Computing Machinery, December 2013. ISBN: 978-1-4503-2638-4. DOI: `10.1145/2540708.2540711`.

[38] L. Lou, P. Nguyen, J. Lawrence and C. Barnes. Image Perforation: Automatically Accelerating Image Pipelines by Intelligently Skipping Samples. *ACM Transactions on Graphics*, 35(5):153:1–153:14, September 2016. ISSN: 0730-0301. DOI: `10.1145/2904903`.

[39] S. Li, S. Park and S. Mahlke. Sculptor: Flexible Approximation with Selective Dynamic Loop Perforation. In *Proceedings of the 2018 International Conference on Supercomputing*, ICS '18, pages 341–351, New York, NY, USA. Association for Computing Machinery, June 2018. ISBN: 978-1-4503-5783-8. DOI: `10.1145/3205289.3205317`.

[40] S. Mitra, M. K. Gupta, S. Misailovic and S. Bagchi. Phase-aware optimization in approximate computing. In *Proceedings of the 2017 International Symposium on Code Generation and Optimization*, CGO '17, pages 185–196, Austin, USA. IEEE Press, February 2017. ISBN: 978-1-5090-4931-8. DOI: `10.1109/CGO.2017.7863739`.

[41] J.-M. Muller. Elementary functions: algorithms and implementation. In USA. Birkhauser Boston, Inc., 1997. ISBN: 978-0-8176-3990-7.

[42] S. Nagayama, T. Sasao and J. T. Butler. Design Method for Numerical Function Generators Based on Polynomial Approximation for FPGA Implementation. In *10th Euromicro Conference on Digital System Design Architectures, Methods and Tools (DSD 2007)*, pages 280–287, August 2007. DOI: `10.1109/DSD.2007.4341481`.

[43] R. Michard, A. Tisserand and N. Veyrat-Charvillon. Small FPGA polynomial approximations with 3-bit coefficients and low-precision estimations of the powers of x. In *2005 IEEE International Conference on Application-Specific Systems, Architecture Processors (ASAP'05)*, pages 334–339, July 2005. DOI: `10.1109/ASAP.2005.59`. ISSN: 1063-6862.

[44] N. Brisebarre, J.-M. Muller and A. Tisserand. Sparse-coefficient polynomial approximations for hardware implementations. In *Conference Record of the Thirty-Eighth Asilomar Conference on Signals, Systems and Computers, 2004*. Volume 1, 532–535 Vol.1, November 2004. DOI: `10.1109/ACSSC.2004.1399189`.

[45] S. Chevillard, M. Joldeş and C. Lauter. Sollya: An Environment for the Development of Numerical Codes. In K. Fukuda, J. v. d. Hoeven, M. Joswig and N. Takayama, editors, *Mathematical Software – ICMS 2010*, Lecture Notes in Computer Science, pages 28–31, Berlin, Heidelberg. Springer, 2010. ISBN: 978-3-642-15582-6. DOI: `10.1007/978-3-642-15582-6_5`.

[46] F. de Dinechin, M. Joldes and B. Pasca. Automatic generation of polynomial-based hardware architectures for function evaluation. In *ASAP 2010 - 21st IEEE International Conference on Application-specific Systems, Architectures and Processors*, pages 216–222, Rennes, France. IEEE, July 2010. DOI: `10.1109/ASAP.2010.5540952`.

[47] O. Kupriianova and C. Lauter. Metalibm: A Mathematical Functions Code Generator. In H. Hong and C. Yap, editors, *Mathematical Software – ICMS 2014*, Lecture Notes in Computer Science, pages 713–717, Berlin, Heidelberg. Springer, 2014. ISBN: 978-3-662-44199-2. DOI: `10.1007/978-3-662-44199-2_106`.

[48] J. Hao, J. Xu, S. Guo and Y. Xia. Design of variable precision transcendental function automatic generator. *The Journal of Supercomputing*, 78(2):2196–2218, February 2022. ISSN: 1573-0484. DOI: `10.1007/s11227-021-03937-8`.

[49] J.-M. Muller. Elementary Functions and Approximate Computing. *Proceedings of the IEEE*, 108(12):2136–2149, December 2020. ISSN: 1558-2256. DOI: `10.1109/JPROC.2020.2991885`.

[50] P.-T. P. Tang. Table-driven implementation of the exponential function in IEEE floating-point arithmetic. *ACM Transactions on Mathematical Software*, 15(2):144–157, June 1989. ISSN: 0098-3500. DOI: `10.1145/63522.214389`.

[51] P.-T. P. Tang. Table-driven implementation of the logarithm function in IEEE floating-point arithmetic. *ACM Transactions on Mathematical Software*, 16(4):378–400, December 1990. ISSN: 0098-3500. DOI: `10.1145/98267.98294`.

[52] P.-T. P. Tang. Table-lookup algorithms for elementary functions and their error analysis. In *[1991] Proceedings 10th IEEE Symposium on Computer Arithmetic*, pages 232–236, June 1991. DOI: `10.1109/ARITH.1991.145565`.

[53] P. T. P. Tang. Table-driven implementation of the Expm1 function in IEEE floating-point arithmetic. *ACM Transactions on Mathematical Software*, 18(2):211–222, June 1992. ISSN: 0098-3500. DOI: `10.1145/146847.146928`.

[54] D. Das Sarma and D. Matula. Faithful bipartite ROM reciprocal tables. In *Proceedings of the 12th Symposium on Computer Arithmetic*, pages 17–28, July 1995. DOI: `10.1109/ARITH.1995.465381`.

[55] M. Schulte and J. Stine. Approximating elementary functions with symmetric bipartite tables. *IEEE Transactions on Computers*, 48(8):842–847, August 1999. ISSN: 1557-9956. DOI: `10.1109/12.795125`.

[56] J. E. Stine and M. J. Schulte. The Symmetric Table Addition Method for Accurate Function Approximation. *Journal of VLSI signal processing systems for signal, image and video technology*, 21(2):167–177, June 1999. ISSN: 0922-5773. DOI: `10.1023/A:1008004523235`.

[57] J.-M. Muller. A Few Results on Table-Based Methods. *Reliable Computing*, 5(3):279–288, August 1999. ISSN: 1573-1340. DOI: `10.1023/A:1009984523264`.

[58] H. Sun, Y. Luo, Y. Ha, Y. Shi, Y. Gao, Q. Shen and H. Pan. A Universal Method of Linear Approximation With Controllable Error for the Efficient Implementation of Transcendental Functions. *IEEE Transactions on Circuits and Systems I: Regular Papers*, 67(1):177–188, January 2020. ISSN: 1558-0806. DOI: `10.1109/TCSI.2019.2939563`.

[59] M. An, Y. Luo, M. Zheng, Y. Wang, H. Dong, Z. Wang, C. Peng and H. Pan. *Piecewise Parabolic Approximate Computation Based on an Error-Flattened Segmenter and a Novel Quantizer*, volume 10 of number 21. Multidisciplinary Digital Publishing Institute, January 2021, page 2704. DOI: `10.3390/electronics10212704`. Number: 21.

[60] S.-F. Hsiao, H.-J. Ko and C.-S. Wen. Two-Level Hardware Function Evaluation Based on Correction of Normalized Piecewise Difference Functions. *IEEE Transactions on Circuits and Systems II: Express Briefs*, 59(5):292–296, May 2012. ISSN: 1558-3791. DOI: `10.1109/TCSII.2012.2190862`.

[61] D. M. Ellaithy, M. A. El-Moursy, G. H. Ibrahim, A. Zaki and A. Zekry. Double Logarithmic Arithmetic Technique for Low-Power 3-D Graphics Applications. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 25(7):2144–2152, July 2017. ISSN: 1557-9999. DOI: `10.1109/TVLSI.2017.2667714`.

[62] D. De Caro, N. Petra and A. G. M. Strollo. Efficient Logarithmic Converters for Digital Signal Processing Applications. *IEEE Transactions on Circuits and Systems II: Express Briefs*, 58(10):667–671, October 2011. ISSN: 1558-3791. DOI: `10.1109/TCSII.2011.2164159`.

[63] B.-G. Nam, H. Kim and H.-J. Yoo. Power and Area-Efficient Unified Computation of Vector and Elementary Functions for Handheld 3D Graphics Systems. *IEEE Transactions on Computers*, 57(4):490–504, April 2008. ISSN: 1557-9956. DOI: `10.1109/TC.2008.12`.

[64] B.-G. Nam and H.-J. Yoo. An Embedded Stream Processor Core Based on Logarithmic Arithmetic for a Low-Power 3-D Graphics SoC. *IEEE Journal of Solid-State Circuits*, 44(5):1554–1570, May 2009. ISSN: 1558-173X. DOI: `10.1109/JSSC.2009.2016698`.

[65] D.-U. Lee, W. Luk, J. Villasenor and P. Cheung. Hierarchical segmentation schemes for function evaluation. In *Proceedings. 2003 IEEE International Conference on Field-Programmable Technology (FPT) (IEEE Cat. No.03EX798)*, pages 92–99, December 2003. DOI: `10.1109/FPT.2003.1275736`.

[66] D.-U. Lee, R. Cheung, W. Luk and J. Villasenor. Hardware Implementation Trade-Offs of Polynomial Approximations and Interpolations. *IEEE Transactions on Computers*, 57(5):686–701, May 2008. ISSN: 1557-9956. DOI: 10.1109/TC.2007.70847.

[67] D.-U. Lee, R. C. C. Cheung, W. Luk and J. D. Villasenor. Hierarchical Segmentation for Hardware Function Evaluation. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 17(1):103–116, January 2009. ISSN: 1557-9999. DOI: 10.1109/TVLSI.2008.2003165.

[68] M. Zhu, Y. Ha, C. Gu and L. Gao. An Optimized Logarithmic Converter With Equal Distribution of Relative Errors. *IEEE Transactions on Circuits and Systems II: Express Briefs*, 63(9):848–852, September 2016. ISSN: 1558-3791. DOI: 10.1109/TCSII.2016.2535041.

[69] C.-W. Liu, S.-H. Ou, K.-C. Chang, T.-C. Lin and S.-K. Chen. A Low-Error, Cost-Efficient Design Procedure for Evaluating Logarithms to Be Used in a Logarithmic Arithmetic Processor. *IEEE Transactions on Computers*, 65(4):1158–1164, April 2016. ISSN: 1557-9956. DOI: 10.1109/TC.2015.2441696.

[70] H. Dong, M. Wang, Y. Luo, M. Zheng, M. An, Y. Ha and H. Pan. PLAC: Piecewise Linear Approximation Computation for All Nonlinear Unary Functions. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 28(9):2014–2027, September 2020. ISSN: 1557-9999. DOI: 10.1109/TVLSI.2020.3004602.

[71] J. Winograd and S. Nawab. Incremental refinement of DFT and STFT approximations. *IEEE Signal Processing Letters*, 2(2):25–27, February 1995. ISSN: 1558-2361. DOI: 10.1109/97.365530.

[72] J. Winograd. Incremental refinement structures for approximate signal processing, 1997.

[73] S. H. Nawab, A. V. Oppenheim, A. P. Chandrakasan, J. M. Winograd and J. T. Ludwig. Approximate Signal Processing. *Journal of VLSI signal processing systems for signal, image and video technology*, 15(1):177–200, January 1997. ISSN: 0922-5773. DOI: 10.1023/A:1007986707921.

[74] Y. Andreopoulos and I. Patras. Incremental Refinement of Image Salient-Point Detection. *IEEE Transactions on Image Processing*, 17(9):1685–1699, September 2008. ISSN: 1941-0042. DOI: 10.1109/TIP.2008.2001051.

[75] A. Roldao-Lopes, A. Shahzad, G. A. Constantinides and E. C. Kerrigan. More Flops or More Precision? Accuracy Parameterizable Linear Equation Solvers for Model Predictive Control. In *2009 17th IEEE Symposium on Field Programmable Custom Computing Machines*, pages 209–216, April 2009. DOI: 10.1109/FCCM.2009.19.

[76] V. K. Chippa, D. Mohapatra, K. Roy, S. T. Chakradhar and A. Raghunathan. Scalable Effort Hardware Design. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 22(9):2004–2016, September 2014. ISSN: 1557-9999. DOI: 10.1109/TVLSI.2013.2276759.

[77] D. S. Khudia, B. Zamirai, M. Samadi and S. Mahlke. Rumba: An online quality management system for approximate computing. In *2015 ACM/IEEE 42nd Annual International Symposium on Computer Architecture (ISCA)*, pages 554–566, June 2015. DOI: 10.1145/2749469.2750371. ISSN: 1063-6897.

[78] Y. Tian, Q. Zhang, T. Wang, F. Yuan and Q. Xu. ApproxMA: Approximate Memory Access for Dynamic Precision Scaling. In *Proceedings of the 25th edition on Great Lakes Symposium on VLSI*, GLSVLSI '15, pages 337–342, New York, NY, USA. Association for Computing Machinery, May 2015. ISBN: 978-1-4503-3474-7. DOI: 10.1145/2742060.2743759.

[79] M. Rinard. Probabilistic accuracy bounds for fault-tolerant computations that discard tasks. In *Proceedings of the 20th annual international conference on Supercomputing*, ICS '06, pages 324–334, New York, NY, USA. Association for Computing Machinery, June 2006. ISBN: 978-1-59593-282-2. DOI: 10.1145/1183401.1183447.

[80] I. Goiri, R. Bianchini, S. Nagarakatte and T. D. Nguyen. ApproxHadoop: Bringing Approximations to MapReduce Frameworks. *ACM SIGPLAN Notices*, 50(4):383–397, March 2015. ISSN: 0362-1340. DOI: 10.1145/2775054.2694351.

[81] S. Byna, J. Meng, A. Raghunathan, S. Chakradhar and S. Cadambi. Best-effort semantic document search on GPUs. In *Proceedings of the 3rd Workshop on General-Purpose Computation on Graphics Processing Units*, GPGPU-3, pages 86–93, New York, NY, USA. Association for Computing Machinery, March 2010. ISBN: 978-1-60558-935-0. DOI: 10.1145/1735688.1735705.

[82] S. T. Chakradhar and A. Raghunathan. Best-effort computing: Re-thinking parallel software and hardware. In *Design Automation Conference*, pages 865–870, June 2010. DOI: 10.1145/1837274.1837492. ISSN: 0738-100X.

[83] A. Raha, S. Venkataramani, V. Raghunathan and A. Raghunathan. Quality configurable reduce-and-rank for energy efficient approximate computing. In *2015 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 665–670, March 2015. DOI: 10.7873/DATE.2015.0569. ISSN: 1558-1101.

[84] V. Vassiliadis, K. Parasyris, C. Chalios, C. D. Antonopoulos, S. Lalis, N. Bellas, H. Vandierendonck and D. S. Nikolopoulos. A programming model and runtime system for significance-aware energy-efficient computing. *ACM SIGPLAN Notices*, 50(8):275–276, January 2015. ISSN: 0362-1340. DOI: 10.1145/2858788.2688546.

[85] S. Agarwal, B. Mozafari, A. Panda, H. Milner, S. Madden and I. Stoica. BlinkDB: queries with bounded errors and bounded response times on very large data. In *Proceedings of the 8th ACM European Conference on Computer Systems*, EuroSys '13, pages 29–42, New York, NY, USA. Association for Computing Machinery, April 2013. ISBN: 978-1-4503-1994-2. DOI: 10.1145/2465351.2465355.

[86] F. Olken and D. Rotem. Random sampling from database files: A survey. In Z. Michalewicz, editor, *Statistical and Scientific Database Management*, Lecture Notes in Computer Science, pages 92–111, Berlin, Heidelberg. Springer, 1990. ISBN: 978-3-540-46968-1. DOI: 10.1007/3-540-52342-1_23.

[87] F. Olken and D. Rotem. Random sampling from databases: a survey. *Statistics and Computing*, 5(1):25–42, March 1995. ISSN: 1573-1375. DOI: 10.1007/BF00140664.

[88] S. Kandula, A. Shanbhag, A. Vitorovic, M. Olma, R. Grandl, S. Chaudhuri and B. Ding. Quickr: lazily approximating complex adhoc queries in bigdata clusters. SIGMOD '16:631–646, 2016. DOI: 10.1145/2882903.2882940.

[89] J. Ansel, Y. L. Wong, C. Chan, M. Olszewski, A. Edelman and S. Amarasinghe. Language and compiler support for auto-tuning variable-accuracy algorithms. In *International Symposium on Code Generation and Optimization (CGO 2011)*, pages 85–96, April 2011. DOI: 10.1109/CGO.2011.5764677.

[90] A. Wieland and R. Leighton. Geometric analysis of neural network capabilities. In *Proceedings of IEEE First International Conference on Neural Networks, San Diego, CA*, 385–392 vol.3, 1987.

[91] Irie and Miyake. Capabilities of three-layered perceptrons. In *IEEE 1988 International Conference on Neural Networks*, 641–648 vol.1, July 1988. DOI: 10.1109/ICNN.1988.23901.

[92] K.-I. Funahashi. On the approximate realization of continuous mappings by neural networks. *Neural Networks*, 2(3):183–192, January 1989. ISSN: 0893-6080. DOI: 10.1016/0893-6080(89)90003-8.

[93] T. Chen and H. Chen. Approximations of continuous functionals by neural networks with application to dynamic systems. *IEEE Transactions on Neural Networks*, 4(6):910–918, November 1993. ISSN: 1941-0093. DOI: 10.1109/72.286886.

[94] K. Hornik. Some new results on neural network approximation. *Neural Networks*, 6(8):1069–1072, January 1993. ISSN: 0893-6080. DOI: 10.1016/S0893-6080(09)80018-X.

[95] R. DeVore, B. Hanin and G. Petrova. Neural network approximation. *Acta Numerica, Cambridge University Press*, 30:327–444, May 2021. ISSN: 0962-4929, 1474-0508. DOI: 10.1017/S0962492921000052.

[96] H. Esmaeilzadeh, A. Sampson, L. Ceze and D. Burger. Neural Acceleration for General-Purpose Approximate Programs. In *2012 45th Annual IEEE/ACM International Symposium on Microarchitecture*, pages 449–460, December 2012. DOI: 10.1109/MICRO.2012.48. ISSN: 2379-3155.

[97] R. St. Amant, A. Yazdanbakhsh, J. Park, B. Thwaites, H. Esmaeilzadeh, A. Hassibi, L. Ceze and D. Burger. General-purpose code acceleration with limited-precision analog computation. *ACM SIGARCH Computer Architecture News*, 42(3):505–516, June 2014. ISSN: 0163-5964. DOI: 10.1145/2678373.2665746.

[98] L. McAfee and K. Olukotun. EMEURO: A framework for generating multi-purpose accelerators via deep learning. In *2015 IEEE/ACM International Symposium on Code Generation and Optimization (CGO)*, pages 125–135, February 2015. DOI: 10.1109/CGO.2015.7054193.

[99] S. Eldridge, F. Raudies, D. Zou and A. Joshi. Neural network-based accelerators for transcendental function approximation. In *Proceedings of the 24th edition of the great lakes symposium on VLSI*, GLSVLSI '14, pages 169–174, New York, NY, USA. Association for Computing Machinery, May 2014. ISBN: 978-1-4503-2816-6. DOI: 10.1145/2591513.2591534.

[100] D. Citron, D. Feitelson and L. Rudolph. Accelerating multi-media processing by implementing memoing in multiplication and division units. *ACM SIGPLAN Notices*, 33(11):252–261, October 1998. ISSN: 0362-1340. DOI: 10.1145/291006.291056.

[101] D. Dubois, H. Prade and F. Sèdes. Fuzzy Logic Techniques in Multimedia Databases Querying: A Preliminary Investigation of the Potentials. In R. Meersman, Z. Tari and S. Stevens, editors, *Database Semantics: Semantic Issues in Multimedia Systems*, IFIP — The International Federation for Information Processing, pages 211–229. Springer US, Boston, MA, 1999. ISBN: 978-0-387-35561-0. DOI: 10.1007/978-0-387-35561-0_13.

[102] C. Alvarez, J. Corbal and M. Valero. Fuzzy memoization for floating-point multimedia applications. *IEEE Transactions on Computers*, 54(7):922–927, July 2005. ISSN: 1557-9956. DOI: 10.1109/TC.2005.119.

[103] Y. Ono and K. Usami. Approximate Computing Technique Using Memoization and Simplified Multiplication. In *2019 34th International Technical Conference on Circuits/Systems, Computers and Communications (ITC-CSCC)*, pages 1–4, June 2019. DOI: 10.1109/ITC-CSCC.2019.8793369.

[104] Y. Fang, H. Li and X. Li. SoftPCM: Enhancing Energy Efficiency and Lifetime of Phase Change Memory in Video Applications via Approximate Write. In *2012 IEEE 21st Asian Test Symposium*, pages 131–136, November 2012. DOI: 10.1109/ATS.2012.57. ISSN: 2377-5386.

[105] A. Ranjan, S. Venkataramani, X. Fong, K. Roy and A. Raghunathan. Approximate storage for energy efficient spintronic memories. In *Proceedings of the 52nd Annual Design Automation Conference*, pages 1–6, San Francisco California. Association for Computing Machinery, June 2015. ISBN: 978-1-4503-3520-1. DOI: 10.1145/2744769.2744799.

[106] A. Sampson, J. Nelson, K. Strauss and L. Ceze. Approximate storage in solid-state memories. In *2013 46th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pages 25–36, December 2013.

[107] S. Ganapathy, G. Karakonstantis, A. Teman and A. Burg. Mitigating the impact of faults in unreliable memories for error-resilient applications. In *2015 52nd ACM/EDAC/IEEE Design Automation Conference (DAC)*, pages 1–6, June 2015. DOI: 10.1145/2744769.2744871. ISSN: 0738-100X.

[108] Q. Zhang, T. Wang, Y. Tian, F. Yuan and Q. Xu. ApproxANN: An approximate computing framework for artificial neural network. In *2015 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 701–706, March 2015. DOI: 10.7873/DATE.2015.0618. ISSN: 1558-1101.

[109] W. Qin and S. Idreos. Adaptive Data Skipping in Main-Memory Systems. In *Proceedings of the 2016 International Conference on Management of Data*, pages 2255–2256, San Francisco California USA. Association for Computing Machinery, June 2016. ISBN: 978-1-4503-3531-7. DOI: 10.1145/2882903.2914836.

[110] S. Liu, K. Pattabiraman, T. Moscibroda and B. G. Zorn. Flikker: saving DRAM refresh-power through critical data partitioning. *ACM SIGPLAN Notices*, 46(3):213–224, March 2011. ISSN: 0362-1340. DOI: 10.1145/1961296.1950391.

[111] K. Cho, Y. Lee, Y. H. Oh, G.-c. Hwang and J. W. Lee. eDRAM-based Tiered-Reliability Memory with applications to low-power frame buffers. In *2014 IEEE/ACM International Symposium on Low Power Electronics and Design (ISLPED)*, pages 333–338, August 2014. DOI: 10.1145/2627369.2627626.

[112] S. Ganapathy, A. Teman, R. Giterman, A. Burg and G. Karakonstantis. Approximate computing with unreliable dynamic memories. In *2015 IEEE 13th International New Circuits and Systems Conference (NEWCAS)*, pages 1–4, June 2015. DOI: 10.1109/NEWCAS.2015.7182027.

[113] A. Raha, S. Sutar, H. Jayakumar and V. Raghunathan. Quality Configurable Approximate DRAM. *IEEE Transactions on Computers*, 66(7):1172–1187, July 2017. ISSN: 1557-9956. DOI: 10.1109/TC.2016.2640296.

[114] I. J. Chang, D. Mohapatra and K. Roy. A Priority-Based 6T/8T Hybrid SRAM Architecture for Aggressive Voltage Scaling in Video Applications. *IEEE Transactions on Circuits and Systems for Video Technology*, 21(2):101–112, February 2011. ISSN: 1558-2205. DOI: 10.1109/TCSVT.2011.2105550.

[115] H. Esmaeilzadeh, A. Sampson, L. Ceze and D. Burger. Architecture support for disciplined approximate programming. In *Proceedings of the seventeenth international conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS XVII, pages 301–312, New York, NY, USA. Association for Computing Machinery, March 2012. ISBN: 978-1-4503-0759-8. DOI: 10.1145/2150976.2151008.

[116] B. Salami, O. S. Unsal and A. Cristal Kestelman. Comprehensive Evaluation of Supply Voltage Underscaling in FPGA on-Chip Memories. In *2018 51st Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pages 724–736, October 2018. DOI: 10.1109/MICRO.2018.00064.

[117] T. Yeh, P. Faloutsos, M. Ercegovac, S. Patel and G. Reinman. The Art of Deception: Adaptive Precision Reduction for Area Efficient Physics Acceleration. In *Proceedings of the 40th Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO 40, pages 394–406, USA. IEEE Computer Society, December 2007. ISBN: 978-0-7695-3047-5. DOI: 10.1109/MICRO.2007.41.

[118] C.-C. Hsiao, S.-L. Chu and C.-Y. Chen. Energy-aware hybrid precision selection framework for mobile GPUs. *Computers & Graphics*, 37(5):431–444, August 2013. ISSN: 0097-8493. DOI: 10.1016/j.cag.2013.03.003.

[119] C. Rubio-González, C. Nguyen, H. D. Nguyen, J. Demmel, W. Kahan, K. Sen, D. H. Bailey, C. Iancu and D. Hough. Precimonious: tuning assistant for floating-point precision. In *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, SC '13, pages 1–12, New York, NY, USA. Association for Computing Machinery, November 2013. ISBN: 978-1-4503-2378-9. DOI: 10.1145/2503210.2503296.

[120] R. Han, J. Demmel and Y. You. Auto-Precision Scaling for Distributed Deep Learning. In B. L. Chamberlain, A.-L. Varbanescu, H. Ltaief and P. Luszczek, editors, *High Performance Computing*, Lecture Notes in Computer Science, pages 79–97, Cham. Springer International Publishing, 2021. ISBN: 978-3-030-78713-4. DOI: 10.1007/978-3-030-78713-4_5.

[121] H. Sim, S. Kenzhegulov and J. Lee. DPS: Dynamic Precision Scaling for Stochastic Computing-based Deep Neural Networks. In *2018 55th ACM/ESDA/IEEE Design Automation Conference (DAC)*, pages 1–6, June 2018. DOI: 10.1109/DAC.2018.8465700.

[122] S. Draghici. On the capabilities of neural networks using limited precision weights. *Neural Networks*, 15(3):395–414, April 2002. ISSN: 0893-6080. DOI: 10.1016/S0893-6080(02)00032-1.

[123] J. H. Ko, J. Fromm, M. Philipose, I. Tashev and S. Zarar. Precision Scaling of Neural Networks for Efficient Audio Processing. Technical report, arXiv, December 2017. DOI: 10.48550/arXiv.1712.01340.

[124] K. Wang, Z. Liu, Y. Lin, J. Lin and S. Han. HAQ: Hardware-Aware Automated Quantization With Mixed Precision. In *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 8604–8612, June 2019. DOI: 10.1109/CVPR.2019.00881. ISSN: 2575-7075.

[125] C. Wu, M. Wang, X. Li, J. Lu, K. Wang and L. He. Phoenix: A Low-Precision Floating-Point Quantization Oriented Architecture for Convolutional Neural Networks, February 2020. DOI: 10.48550/arXiv.2003.02628.

[126] L. Zhang, Y. Peng, X. Hu, A. Huang and T. Tian. FPGA-Based Multi-precision Architecture for Accelerating Large-Scale Floating-Point Matrix Computing. In *Network and Parallel Computing: 17th IFIP WG 10.3 International Conference, NPC 2020, Zhengzhou, China, September 28–30, 2020, Revised Selected Papers*, pages 191–202, Berlin, Heidelberg. Springer-Verlag, September 2020. ISBN: 978-3-030-79477-4. DOI: 10.1007/978-3-030-79478-1_17.

[127] J. d. F. Licht, C. A. Pattison, A. N. Ziogas, D. Simmons-Duffin and T. Hoefler. Fast Arbitrary Precision Floating Point on FPGA. In pages 1–9. IEEE Computer Society, May 2022. ISBN: 978-1-66548-332-2. DOI: 10.1109/FCCM53951.2022.9786219.

[128] J. G. Pandey, A. Karmakar, C. Shekhar and S. Gurunarayanan. An FPGA-based fixed-point architecture for binary logarithmic computation. In *2013 IEEE Second International Conference on Image Information Processing (ICIIP-2013)*, pages 383–388, December 2013. DOI: 10.1109/ICIIP.2013.6707620.

[129] H. Jiang, J. Han and F. Lombardi. A Comparative Review and Evaluation of Approximate Adders. In *Proceedings of the 25th edition on Great Lakes Symposium on VLSI*, pages 343–348, Pittsburgh Pennsylvania USA. Association for Computing Machinery, May 2015. ISBN: 978-1-4503-3474-7. DOI: 10.1145/2742060.2743760.

[130] V. Gupta, D. Mohapatra, S. P. Park, A. Raghunathan and K. Roy. IMPACT: IMPrecise adders for low-power approximate computing. In *IEEE/ACM International Symposium on Low Power Electronics and Design*, pages 409–414, August 2011. DOI: 10.1109/ISLPED.2011.5993675.

[131] J. M. Rabaey. *Digital Integrated Circuits: A Design Perspective.* Prentice-Hall, Inc., USA, 1996. ISBN: 0131786091.

[132] V. Gupta, D. Mohapatra, A. Raghunathan and K. Roy. Low-Power Digital Signal Processing Using Approximate Adders. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 32(1):124–137, January 2013. ISSN: 1937-4151. DOI: 10.1109/TCAD.2012.2217962.

[133] Z. Yang, A. Jain, J. Liang, J. Han and F. Lombardi. Approximate XOR/XNOR-based adders for inexact computing. In *2013 13th IEEE International Conference on Nanotechnology (IEEE-NANO 2013)*, pages 690–693, August 2013. DOI: 10.1109/NANO.2013.6720793. ISSN: 1944-9399.

[134] D. Nanu, R. P. K, D. Sowkarthiga and K. S. A. Ameen. Approximate Adder Design Using CPL Logic for Image Compression. *International Journal of Innovative Research and Development*, 0(0), May 2014. ISSN: 2278-0211. Number: 0.

[135] A. Gogoi and V. Kumar. Design of low power, area efficient and high speed approximate adders for inexact computing. In *2016 International Conference on Signal Processing and Communication (ICSC)*, pages 452–456, December 2016. DOI: 10.1109/ICSPCom.2016.7980623.

[136] H. A. Almurib, T. N. Kumar and F. Lombardi. Inexact designs for approximate low power addition by cell replacement. In *2016 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 660–665, March 2016. ISSN: 1558-1101.

[137] B. S. Prabakaran, S. Rehman, M. A. Hanif, S. Ullah, G. Mazaheri, A. Kumar and M. Shafique. DeMAS: An efficient design methodology for building approximate adders for FPGA-based systems. In *2018 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 917–920, March 2018. DOI: 10.23919/DATE.2018.8342140. ISSN: 1558-1101.

[138] K. S. Babu and N. Balaji. Approximation of digital circuits using cartesian genetic programming. In *2016 International Conference on Communication and Electronics Systems (ICCES)*, pages 1–6, October 2016. DOI: 10.1109/CESYS.2016.7889978.

[139] N. Zhu, W. L. Goh, W. Zhang, K. S. Yeo and Z. H. Kong. Design of Low-Power High-Speed Truncation-Error-Tolerant Adder and Its Application in Digital Signal Processing. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 18(8):1225–1229, August 2010. ISSN: 1557-9999. DOI: 10.1109/TVLSI.2009.2020591.

[140] D. Celia, V. Vasudevan and N. Chandrachoodan. Optimizing power-accuracy trade-off in approximate adders. In *2018 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 1488–1491, March 2018. DOI: 10.23919/DATE.2018.8342248. ISSN: 1558-1101.

[141] H. R. Mahdiani, A. Ahmadi, S. M. Fakhraie and C. Lucas. Bio-Inspired Imprecise Computational Blocks for Efficient VLSI Implementation of Soft-Computing Applications. *IEEE Transactions on Circuits and Systems I: Regular Papers*, 57(4):850–862, April 2010. ISSN: 1558-0806. DOI: 10.1109/TCSI.2009.2027626.

[142] P. Albicocco, G. C. Cardarilli, A. Nannarelli, M. Petricca and M. Re. Imprecise arithmetic for low power image processing. In *2012 Conference Record of the Forty Sixth Asilomar Conference on Signals, Systems and Computers (ASILOMAR)*, pages 983–987, November 2012. DOI: `10.1109/ACSSC.2012.6489164`. ISSN: 1058-6393.

[143] A. Dalloo, A. Najafi and A. Garcia-Ortiz. Systematic Design of an Approximate Adder: The Optimized Lower Part Constant-OR Adder. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 26(8):1595–1599, August 2018. ISSN: 1557-9999. DOI: `10.1109/TVLSI.2018.2822278`.

[144] P. Balasubramanian, R. Nayar, D. L. Maskell and N. E. Mastorakis. An Approximate Adder With a Near-Normal Error Distribution: Design, Error Analysis and Practical Application. *IEEE Access*, 9:4518–4530, 2021. ISSN: 2169-3536. DOI: `10.1109/ACCESS.2020.3047651`.

[145] J. Echavarria, S. Wildermann, A. Becher, J. Teich and D. Ziener. FAU: Fast and error-optimized approximate adder units on LUT-Based FPGAs. In *2016 International Conference on Field-Programmable Technology (FPT)*, pages 213–216, December 2016. DOI: `10.1109/FPT.2016.7929536`.

[146] P. Balasubramanian and D. Maskell. Hardware Efficient Approximate Adder Design. In *TENCON 2018 - 2018 IEEE Region 10 Conference*, pages 0806–0810, October 2018. DOI: `10.1109/TENCON.2018.8650127`. ISSN: 2159-3450.

[147] A. Becher, J. Echavarria, D. Ziener, S. Wildermann and J. Teich. A LUT-Based Approximate Adder. In *2016 IEEE 24th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, pages 27–27, May 2016. DOI: `10.1109/FCCM.2016.16`.

[148] D. Mohapatra, V. K. Chippa, A. Raghunathan and K. Roy. Design of voltage-scalable meta-functions for approximate computing. In *2011 Design, Automation & Test in Europe*, pages 1–6, March 2011. DOI: `10.1109/DATE.2011.5763154`. ISSN: 1558-1101.

[149] A. B. Kahng and S. Kang. Accuracy-configurable adder for approximate arithmetic designs. In *Proceedings of the 49th Annual Design Automation Conference on - DAC '12*, page 820, San Francisco, California. ACM Press, 2012. ISBN: 978-1-4503-1199-1. DOI: `10.1145/2228360.2228509`.

[150] R. Ye, T. Wang, F. Yuan, R. Kumar and Q. Xu. On reconfiguration-oriented approximate adder design and its application. In *2013 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pages 48–54, November 2013. DOI: `10.1109/ICCAD.2013.6691096`. ISSN: 1558-2434.

[151] M. Shafique, W. Ahmad, R. Hafiz and J. Henkel. A low latency generic accuracy configurable adder. In *Proceedings of the 52nd Annual Design Automation Conference*, pages 1–6, San Francisco California. Association for Computing Machinery, June 2015. ISBN: 978-1-4503-3520-1. DOI: `10.1145/2744769.2744778`.

[152] M. A. Hanif, R. Hafiz, O. Hasan and M. Shafique. QuAd: Design and analysis of Quality-area optimal Low-Latency approximate Adders. In *2017 54th ACM/EDAC/IEEE Design Automation Conference (DAC)*, pages 1–6, June 2017. DOI: `10.1145/3061639.3062306`.

[153] K. Du, P. Varman and K. Mohanram. High performance reliable variable latency carry select addition. In *2012 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 1257–1262, March 2012. DOI: 10.1109/DATE.2012.6176685. ISSN: 1558-1101.

[154] Y. Kim, Y. Zhang and P. Li. An energy efficient approximate adder with carry skip for error resilient neuromorphic VLSI systems. In *2013 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pages 130–137, November 2013. DOI: 10.1109/ICCAD.2013.6691108. ISSN: 1558-2434.

[155] I.-C. Lin, Y.-M. Yang and C.-C. Lin. High-Performance Low-Power Carry Speculative Addition With Variable Latency. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 23(9):1591–1603, September 2015. ISSN: 1557-9999. DOI: 10.1109/TVLSI.2014.2355217.

[156] L. Li and H. Zhou. On error modeling and analysis of approximate adders. In *2014 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pages 511–518, November 2014. DOI: 10.1109/ICCAD.2014.7001399. ISSN: 1558-2434.

[157] J. Hu and W. Qian. A new approximate adder with low relative error and correct sign calculation. In *2015 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 1449–1454, March 2015. ISSN: 1558-1101.

[158] S. Ullah, S. S. Sahoo, N. Ahmed, D. Chaudhury and A. Kumar. Appaxo: designing application-specific approximate operators for fpga-based embedded systems. *ACM Trans. Embed. Comput. Syst.*, 21(3), May 2022. ISSN: 1539-9087. DOI: 10.1145/3513262.

[159] V. Mrazek, R. Hrbacek, Z. Vasicek and L. Sekanina. EvoApprox8b: Library of Approximate Adders and Multipliers for Circuit Design and Benchmarking of Approximation Methods. In *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2017*, pages 258–261, Lausanne. IEEE, March 2017. ISBN: 978-3-9815370-8-6. DOI: 10.23919/DATE.2017.7926993.

[160] V. Mrazek, L. Sekanina and Z. Vasicek. Libraries of Approximate Circuits: Automated Design and Application in CNN Accelerators. *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, 10(4):406–418, December 2020. ISSN: 2156-3365. DOI: 10.1109/JETCAS.2020.3032495.

[161] P. Kulkarni, P. Gupta and M. Ercegovac. Trading Accuracy for Power with an Underdesigned Multiplier Architecture. In *Proceedings of the 2011 24th International Conference on VLSI Design*, VLSID '11, pages 346–351, USA. IEEE Computer Society, January 2011. ISBN: 978-0-7695-4348-2. DOI: 10.1109/VLSID.2011.51.

[162] M. Vasudevan and C. Chakrabarti. Image processing using approximate datapath units. In *2014 IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 1544–1547, June 2014. DOI: 10.1109/ISCAS.2014.6865442. ISSN: 2158-1525.

[163] S. Ullah, S. Rehman, B. S. Prabakaran, F. Kriebel, M. A. Hanif, M. Shafique and A. Kumar. Area-Optimized Low-Latency Approximate Multipliers for FPGA-based Hardware Accelerators. In *2018 55th ACM/ESDA/IEEE Design Automation Conference (DAC)*, pages 1–6, June 2018. DOI: 10.1109/DAC.2018.8465781.

[164] N. H. E. Weste and D. M. Harris. *CMOS VLSI design: a circuits and systems perspective*. Addison-Wesley Publishing Company, Boston, 4th ed edition, 2011. ISBN: 978-0-321-54774-3. OCLC: ocn473447233.

[165] F. Farshchi, M. S. Abrishami and S. M. Fakhraie. New approximate multiplier for low power digital signal processing. In *The 17th CSI International Symposium on Computer Architecture & Digital Systems (CADS 2013)*, pages 25–30, October 2013. DOI: `10.1109/CADS.2013.6714233`. ISSN: 2325-937X.

[166] K. Y. Kyaw, W. L. Goh and K. S. Yeo. Low-power high-speed multiplier for error-tolerant application. In *2010 IEEE International Conference of Electron Devices and Solid-State Circuits (EDSSC)*, pages 1–4, December 2010. DOI: `10.1109/EDSSC.2010.5713751`.

[167] S. Narayanamoorthy, H. A. Moghaddam, Z. Liu, T. Park and N. S. Kim. Energy-Efficient Approximate Multiplication for Digital Signal Processing and Classification Applications. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 23(6):1180–1184, June 2015. ISSN: 1557-9999. DOI: `10.1109/TVLSI.2014.2333366`.

[168] K. Bhardwaj and P. S. Mane. ACMA: Accuracy-configurable multiplier architecture for error-resilient System-on-Chip. In *2013 8th International Workshop on Reconfigurable and Communication-Centric Systems-on-Chip (ReCoSoC)*, pages 1–6, July 2013. DOI: `10.1109/ReCoSoC.2013.6581532`.

[169] S. Hashemi, R. I. Bahar and S. Reda. DRUM: A Dynamic Range Unbiased Multiplier for approximate applications. In *2015 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pages 418–425, November 2015. DOI: `10.1109/ICCAD.2015.7372600`.

[170] S. Vahdat, M. Kamal, A. Afzali-Kusha and M. Pedram. TOSAM: An Energy-Efficient Truncation- and Rounding-Based Scalable Approximate Multiplier. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 27(5):1161–1173, May 2019. ISSN: 1557-9999. DOI: `10.1109/TVLSI.2018.2890712`.

[171] C. H. Lin and I.-C. Lin. *High accuracy approximate multiplier with error correction*. IEEE Computer Society, January 2013, pages 33–38. DOI: `10.1109/ICCD.2013.6657022`.

[172] A. Momeni, J. Han, P. Montuschi and F. Lombardi. Design and Analysis of Approximate Compressors for Multiplication. *IEEE Transactions on Computers*, 64:984–994, April 2015. ISSN: 1557-9956. DOI: `10.1109/TC.2014.2308214`.

[173] M. S. Ansari, H. Jiang, B. F. Cockburn and J. Han. Low-Power Approximate Multipliers Using Encoded Partial Products and Approximate Compressors. *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, 8(3):404–416, September 2018. ISSN: 2156-3357, 2156-3365. DOI: `10.1109/JETCAS.2018.2832204`.

[174] A. G. M. Strollo, E. Napoli, D. De Caro, N. Petra and G. D. Meo. Comparison and Extension of Approximate 4-2 Compressors for Low-Power Approximate Multipliers. *IEEE Transactions on Circuits and Systems I: Regular Papers*, 67(9):3021–3034, September 2020. ISSN: 1558-0806. DOI: `10.1109/TCSI.2020.2988353`.

[175] R. Zendegani, M. Kamal, M. Bahadori, A. Afzali-Kusha and M. Pedram. RoBA Multiplier: A Rounding-Based Approximate Multiplier for High-Speed yet Energy-Efficient Digital Signal Processing. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 25(2):393–401, February 2017. ISSN: 1557-9999. DOI: 10.1109/TVLSI.2016.2587696.

[176] P. M. Kogge and H. S. Stone. A parallel algorithm for the efficient solution of a general class of recurrence equations. *IEEE Trans. Comput.*, 22(8):786–793, August 1973. ISSN: 0018-9340. DOI: 10.1109/TC.1973.5009159.

[177] B. Garg and S. Patel. Reconfigurable Rounding Based Approximate Multiplier for Energy Efficient Multimedia Applications. *Wireless Personal Communications*, 118(2):919–931, May 2021. ISSN: 0929-6212, 1572-834X. DOI: 10.1007/s11277-020-08051-1.

[178] J. N. Mitchell. Computer Multiplication and Division Using Binary Logarithms. *IRE Transactions on Electronic Computers*, EC-11(4):512–517, August 1962. ISSN: 0367-9950. DOI: 10.1109/TEC.1962.5219391.

[179] M. S. Kim, A. A. D. Barrio, L. T. Oliveira, R. Hermida and N. Bagherzadeh. Efficient Mitchell's Approximate Log Multipliers for Convolutional Neural Networks. *IEEE Transactions on Computers*, 68(5):660–675, May 2019. ISSN: 1557-9956. DOI: 10.1109/TC.2018.2880742.

[180] D. Mclaren. Improved Mitchell-based logarithmic multiplier for low-power DSP applications. In *IEEE International [Systems-on-Chip] SOC Conference, 2003. Proceedings.* Pages 53–56, September 2003. DOI: 10.1109/SOC.2003.1241461.

[181] H. Saadat, H. Bokhari and S. Parameswaran. Minimally biased multipliers for approximate integer and floating-point multiplication. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 37(11):2623–2635, 2018. DOI: 10.1109/TCAD.2018.2857262.

[182] H. Saadat, H. Javaid, A. Ignjatovic and S. Parameswaran. Realm: reduced-error approximate log-based integer multiplier:1366–1371, 2020. DOI: 10.23919/DATE48585.2020.9116315.

[183] M. S. Ansari, B. F. Cockburn and J. Han. An Improved Logarithmic Multiplier for Energy-Efficient Neural Computing. *IEEE Transactions on Computers*, 70(4):614–625, April 2021. ISSN: 1557-9956. DOI: 10.1109/TC.2020.2992113.

[184] W. Liu, J. Xu, D. Wang, C. Wang, P. Montuschi and F. Lombardi. Design and Evaluation of Approximate Logarithmic Multipliers for Low Power Error-Tolerant Applications. *IEEE Transactions on Circuits and Systems I: Regular Papers*, 65(9):2856–2868, September 2018. ISSN: 1558-0806. DOI: 10.1109/TCSI.2018.2792902.

[185] Z. Ebrahimi, S. Ullah and A. Kumar. LeAp: Leading-one Detection-based Softcore Approximate Multipliers with Tunable Accuracy. In *2020 25th Asia and South Pacific Design Automation Conference (ASP-DAC)*, pages 605–610, January 2020. DOI: 10.1109/ASP-DAC47756.2020.9045171. ISSN: 2153-697X.

[186] L. Chen, J. Han, W. Liu and F. Lombardi. Design of Approximate Unsigned Integer Non-restoring Divider for Inexact Computing. In *Proceedings of the 25th edition on Great Lakes Symposium on VLSI*, GLSVLSI '15, pages 51–56, New York, NY, USA. Association for Computing Machinery, May 2015. ISBN: 978-1-4503-3474-7. DOI: 10.1145/2742060.2742063.

[187] L. Chen, J. Han, W. Liu and F. Lombardi. On the Design of Approximate Restoring Dividers for Error-Tolerant Applications. *IEEE Transactions on Computers*, 65(8):2522–2533, August 2016. ISSN: 1557-9956. DOI: 10.1109/TC.2015.2494005.

[188] S. Hashemi, R. I. Bahar and S. Reda. A low-power dynamic divider for approximate applications. In *Proceedings of the 53rd Annual Design Automation Conference*, pages 1–6, Austin Texas. Association for Computing Machinery, June 2016. ISBN: 978-1-4503-4236-0. DOI: 10.1145/2897937.2897965.

[189] H. Saadat, H. Javaid and S. Parameswaran. Approximate integer and floating-point dividers with near-zero error bias. In *Proceedings of the 56th Annual Design Automation Conference 2019*, DAC '19, Las Vegas, NV, USA. Association for Computing Machinery, June 2019. ISBN: 9781450367257. DOI: 10.1145/3316781.3317773.

[190] J. Y. L. Low and C. C. Jong. Non-iterative high speed division computation based on Mitchell logarithmic method. In *2013 IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 2219–2222, May 2013. DOI: 10.1109/ISCAS.2013.6572317. ISSN: 2158-1525.

[191] W. Liu, J. Li, T. Xu, C. Wang, P. Montuschi and F. Lombardi. Combining Restoring Array and Logarithmic Dividers into an Approximate Hybrid Design. In *2018 IEEE 25th Symposium on Computer Arithmetic (ARITH)*, pages 92–98, June 2018. DOI: 10.1109/ARITH.2018.8464807. ISSN: 2576-2265.

[192] R. Zendegani, M. Kamal, A. Fayyazi, A. Afzali-Kusha, S. Safari and M. Pedram. SEERAD: A high speed yet energy-efficient rounding-based approximate divider. In *2016 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 1481–1484, March 2016. ISSN: 1558-1101.

[193] M. Imani, R. Garcia, A. Huang and T. Rosing. CADE: Configurable Approximate Divider for Energy Efficiency. In *2019 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 586–589, March 2019. DOI: 10.23919/DATE.2019.8715112. ISSN: 1558-1101.

[194] R. Hegde and N. Shanbhag. Soft digital signal processing. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 9(6):813–823, December 2001. ISSN: 1063-8210, 1557-9999. DOI: 10.1109/92.974895.

[195] D. Markovic, V. Stojanovic, B. Nikolic, M. Horowitz and R. Brodersen. Methods for true energy-performance optimization. *IEEE Journal of Solid-State Circuits*, 39(8):1282–1293, August 2004. ISSN: 0018-9200. DOI: 10.1109/JSSC.2004.831796.

[196] A. B. Kahng, S. Kang, R. Kumar and J. Sartori. Slack redistribution for graceful degradation under voltage overscaling. In *2010 15th Asia and South Pacific Design Automation Conference (ASP-DAC)*, pages 825–831, January 2010. DOI: 10.1109/ASPDAC.2010.5419690. ISSN: 2153-697X.

[197] G. Zervakis, F. Ntouskas, S. Xydis, D. Soudris and K. Pekmestzi. VOSsim: A Framework for Enabling Fast Voltage Overscaling Simulation for Approximate Computing Circuits. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 26(6):1204–1208, June 2018. ISSN: 1557-9999. DOI: 10.1109/TVLSI.2018.2803202.

[198] S. Venkataramani, A. Sabne, V. Kozhikkottu, K. Roy and A. Raghunathan. SALSA: Systematic logic synthesis of approximate circuits. In *DAC Design Automation Conference 2012*, pages 796–801, June 2012. DOI: 10.1145/2228360.2228504.

[199] A. Ranjan, A. Raha, S. Venkataramani, K. Roy and A. Raghunathan. ASLAN: Synthesis of approximate sequential circuits. In *2014 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 1–6, March 2014. DOI: 10.7873/DATE.2014.377. ISSN: 1558-1101.

[200] J. F. Miller and P. Thomson. Cartesian Genetic Programming. In R. Poli, W. Banzhaf, W. B. Langdon, J. Miller, P. Nordin and T. C. Fogarty, editors, *Genetic Programming*, Lecture Notes in Computer Science, pages 121–132, Berlin, Heidelberg. Springer, 2000. ISBN: 978-3-540-46239-2. DOI: 10.1007/978-3-540-46239-2_9.

[201] L. Sekanina and Z. Vasicek. Evolutionary computing in approximate circuit design and optimization. In N. Bellas, G. Karakonstantis and C. Bekas, editors, *1st Workshop On Approximate Computing, WAPCO 2015*, pages 1–6, Amsterdam, Holland, January 2015.

[202] Z. Vasicek and L. Sekanina. Evolutionary Approach to Approximate Digital Circuits Design. In volume 19 of number 3, pages 432–444, June 2015. DOI: 10.1109/TEVC.2014.2336175.

[203] Z. Vasicek and L. Sekanina. Evolutionary design of approximate multipliers under different error metrics. In *17th International Symposium on Design and Diagnostics of Electronic Circuits & Systems*, pages 135–140, April 2014. DOI: 10.1109/DDECS.2014.6868777.

[204] F. Vaverka, R. Hrbacek and L. Sekanina. Evolving component library for approximate high level synthesis. In *2016 IEEE Symposium Series on Computational Intelligence (SSCI)*, pages 1–8, December 2016. DOI: 10.1109/SSCI.2016.7850168.

[205] Z. Vasicek, V. Mrazek and L. Sekanina. Evolutionary functional approximation of circuits implemented into FPGAs. In *2016 IEEE Symposium Series on Computational Intelligence (SSCI)*, pages 1–8, December 2016. DOI: 10.1109/SSCI.2016.7850173.

[206] Z. Vasicek and L. Sekanina. Search-based synthesis of approximate circuits implemented into FPGAs. In *2016 26th International Conference on Field Programmable Logic and Applications (FPL)*, pages 1–4, Lausanne, Switzerland. IEEE, August 2016. ISBN: 978-2-8399-1844-2. DOI: 10.1109/FPL.2016.7577305.

[207] A. Lotfi, A. Rahimi, A. Yazdanbakhsh, H. Esmaeilzadeh and R. K. Gupta. Grater: An approximation workflow for exploiting data-level parallelism in FPGA acceleration. In *2016 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 1279–1284, March 2016. DOI: 10.3850/9783981537079_0805. ISSN: 1558-1101.

[208] G. Liu and Z. Zhang. Statistically certified approximate logic synthesis. In *Proceedings of the 36th International Conference on Computer-Aided Design*, ICCAD '17, pages 344–351, Irvine, California. IEEE Press, November 2017. DOI: 10.1109/ICCAD.2017.8203798.

[209] V. Mrazek, M. A. Hanif, Z. Vasicek, L. Sekanina and M. Shafique. autoAx: An Automatic Design Space Exploration and Circuit Building Methodology utilizing Libraries of Approximate Components. In *Proceedings of the 56th Annual Design Automation Conference 2019*, DAC '19, pages 1–6, New York, NY, USA. Association for Computing Machinery, June 2019. ISBN: 978-1-4503-6725-7. DOI: 10.1145/3316781.3317781.

[210] L. Witschen, M. Awais, H. Ghasemzadeh Mohammadi, T. Wiersema and M. Platzner. CIRCA: Towards a modular and extensible framework for approximate circuit generation. *Microelectronics Reliability*, 99:277–290, August 2019. ISSN: 00262714. DOI: 10.1016/j.microrel.2019.04.003.

[211] B. S. Prabakaran, V. Mrazek, Z. Vasicek, L. Sekanina and M. Shafique. ApproxFPGAs: embracing ASIC-based approximate arithmetic components for FPGA-based systems. In *Proceedings of the 57th ACM/EDAC/IEEE Design Automation Conference*, DAC '20, pages 1–6, Virtual Event, USA. IEEE Press, July 2020. ISBN: 978-1-4503-6725-7.

[212] J. Castro-Godínez, J. Mateus-Vargas, M. Shafique and J. Henkel. AxHLS: design space exploration and high-level synthesis of approximate accelerators using approximate functional units and analytical models. In *Proceedings of the 39th International Conference on Computer-Aided Design*, pages 1–9, Virtual Event USA. Association for Computing Machinery, November 2020. ISBN: 978-1-4503-8026-3. DOI: 10.1145/3400302.3415732.

[213] S. Barone, M. Traiola, M. Barbareschi and A. Bosio. Multi-Objective Application-Driven Approximate Design Method. *IEEE Access*, 9:86975–86993, 2021. ISSN: 2169-3536. DOI: 10.1109/ACCESS.2021.3087858.

[214] A. A. Hagberg, D. A. Schult and P. J. Swart. Exploring network structure, dynamics, and function using networkx. In G. Varoquaux, T. Vaught and J. Millman, editors, *Proceedings of the 7th Python in Science Conference*, pages 11–15, Pasadena, CA USA, 2008.

[215] S. Ullah, S. S. Murthy and A. Kumar. SMApproxLib: Library of FPGA-based Approximate Multipliers. In *2018 55th ACM/ESDA/IEEE Design Automation Conference (DAC)*, pages 1–6, June 2018. DOI: 10.1109/DAC.2018.8465845.

[216] S. Conrady. Approximate computing for motion picture camera processing. *Doctoral dissertation, Technical University of Munich (TUM)*, 2023. Submitted.

[217] M. Makni, S. Niar, M. Baklouti and M. Abid. HAPE: A high-level area-power estimation framework for FPGA-based accelerators. *Microprocessors and Microsystems*, 63:11–27, November 2018. ISSN: 01419331. DOI: 10.1016/j.micpro.2018.08.004.

[218] Intel Early Power Estimator. URL: https://www.intel.com/content/www/us/en/programmable/support/support-resources/operation-and-testing/power/pow-powerplay.html (visited on 04/04/2023).

[219] Xilinx Power Estimator. URL: https://www.xilinx.com/products/technology/power/xpe.html (visited on 04/04/2023).

[220] Intel Power Analyzer. URL: https://www.intel.com/content/www/us/en/programmable/support/support-resources/operation-and-testing/power/sof-qts-power.html (visited on 04/04/2023).

[221] ISO/CIE 11664-4:2019. Colorimetry — Part 4: CIE 1976 L*a*b* colour space. Standard, International Organization for Standardization, Geneva, CH, July 2019.

[222] Z. Wang, A. C. Bovik, H. R. Sheikh and E. P. Simoncelli. Image quality assessment: from error visibility to structural similarity. *IEEE Transactions on Image Processing*, 13(4):600–612, April 2004. ISSN: 1057-7149. DOI: 10.1109/TIP.2003.819861.

[223] F.-A. Fortin, F.-M. De Rainville, M.-A. Gardner, M. Parizeau and C. Gagné. DEAP: evolutionary algorithms made easy. *Journal of Machine Learning Research*, 13:2171–2175, July 2012.

[224] D. E. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley Longman Publishing Co., Inc., USA, 1st edition, 1989. ISBN: 978-0-201-15767-3.

[225] K. Deb, S. Agrawal, A. Pratap and T. Meyarivan. A Fast Elitist Non-dominated Sorting Genetic Algorithm for Multi-objective Optimization: NSGA-II. In M. Schoenauer, K. Deb, G. Rudolph, X. Yao, E. Lutton, J. J. Merelo and H.-P. Schwefel, editors, *Parallel Problem Solving from Nature PPSN VI*, Lecture Notes in Computer Science, pages 849–858, Berlin, Heidelberg. Springer, 2000. ISBN: 978-3-540-45356-7. DOI: 10.1007/3-540-45356-3_83.

[226] J. H. Holland. *Adaptation in Natural and Artificial Systems*. University of Michigan Press, Ann Arbor, MI, 1975. ISBN: 9780262581110. DOI: 10.7551/mitpress/1090.001.0001. second edition, 1992.

[227] E. Zitzler, K. Deb and L. Thiele. Comparison of Multiobjective Evolutionary Algorithms: Empirical Results. *Evolutionary Computation*, 8(2):173–195, June 2000. ISSN: 1063-6560. DOI: 10.1162/106365600568202.

[228] J. Kennedy and R. Eberhart. Particle swarm optimization. In *Proceedings of ICNN'95 - International Conference on Neural Networks*, volume 4, 1942–1948 vol.4, November 1995. DOI: 10.1109/ICNN.1995.488968.

[229] S. Kirkpatrick, C. D. Gelatt and M. P. Vecchi. *Optimization by Simulated Annealing*, volume 220 of number 4598. American Association for the Advancement of Science, May 1983, pages 671–680. DOI: 10.1126/science.220.4598.671.

[230] E.-G. Talbi. *Metaheuristics: From Design to Implementation*. John Wiley & Sons, Cologne, Germany, May 2009. ISBN: 978-0-470-49690-9.

[231] K. Deb, A. Pratap, S. Agarwal and T. Meyarivan. A fast and elitist multiobjective genetic algorithm: nsga-ii. *IEEE Transactions on Evolutionary Computation*, 6(2):182–197, 2002. DOI: 10.1109/4235.996017.

[232] A. Hassanat, K. Almohammadi, E. Alkafaween, E. Abunawas, A. Hammouri and V. B. S. Prasath. Choosing Mutation and Crossover Ratios for Genetic Algorithms—A Review with a New Dynamic Approach. In volume 10 of number 12, page 390. Multidisciplinary Digital Publishing Institute, December 2019. DOI: 10.3390/info10120390.

[233] J. Jaskólski. Adaptive hyperparameters for multi-objective genetic algorithms. *Master Thesis, Chair of Integrated Systems, TUM School of Computation, Information and Technology, Technical University of Munich*, November 2022.

[234] A. Kumar, B. Liu, R. Miikkulainen and P. Stone. Effective mutation rate adaptation through group elite selection. In *Proceedings of the Genetic and Evolutionary Computation Conference*, GECCO '22, pages 721–729, Boston, Massachusetts. Association for Computing Machinery, 2022. ISBN: 9781450392372. DOI: `10.1145/3512290.3528706`.

[235] Intel Arria 10 FPGAs. URL: `https://www.intel.com/content/www/us/en/products/details/fpga/arria/10.html` (visited on 04/04/2023).

[236] Intel Quartus Prime. URL: `https://www.intel.com/content/www/us/en/products/details/fpga/development-tools/quartus-prime.html` (visited on 04/04/2023).

[237] Intel Quartus Prime Timing Analyzer. URL: `https://www.intel.com/content/www/us/en/support/programmable/support-resources/design-examples/quartus/sof-qts-timinganalyzer.html` (visited on 04/04/2023).

[238] S. Andriani, H. Brendel, T. Seybold and J. Goldstone. Beyond the Kodak image set: A new reference set of color image sequences. In *2013 IEEE International Conference on Image Processing*, pages 2289–2293. IEEE, September 2013. DOI: `10.1109/ICIP.2013.6738472`.

[239] E. Reinhard and K. Devlin. Dynamic range reduction inspired by photoreceptor physiology. *IEEE Transactions on Visualization and Computer Graphics*, 11(1):13–24, January 2005. ISSN: 1077-2626. DOI: `10.1109/TVCG.2005.9`.

[240] IEC 61966-2-1:1999/AMD1:2003. Multimedia Systems and Equipment - Colour Measurement and Management Part 2-1: Colour Management - Default RGB Colour Space - sRGB - Amendment 1. Standard, International Electrotechnical Commission, Geneva, CH, January 2003.

[241] M. Stokes, M. D. Fairchild and R. S. Berns. Precision requirements for digital color reproduction. *ACM Trans. Graph.*, 11(4):406–422, October 1992. ISSN: 0730-0301. DOI: `10.1145/146443.146482`.

[242] K. Deb and J. Sundar. Reference point based multi-objective optimization using evolutionary algorithms. In *Proceedings of the 8th annual conference on Genetic and evolutionary computation*, GECCO '06, pages 635–642, New York, NY, USA. Association for Computing Machinery, 8th July 2006. ISBN: 978-1-59593-186-3. DOI: `10.1145/1143997.1144112`.

[243] M. Elarbi, S. Bechikh, L. Ben Said and R. Datta. Multi-objective Optimization: Classical and Evolutionary Approaches. In S. Bechikh, R. Datta and A. Gupta, editors, *Recent Advances in Evolutionary Multi-objective Optimization*, Adaptation, Learning, and Optimization, pages 1–30. Springer International Publishing, Cham, Switzerland, 2017. ISBN: 978-3-319-42978-6. DOI: `10.1007/978-3-319-42978-6_1`.

[244] S. Eppe, M. López-Ibáñez, T. Stützle and Y. De Smet. An experimental study of preference model integration into multi-objective optimization heuristics. In *2011 IEEE Congress of Evolutionary Computation (CEC)*, pages 2751–2758, New Orleans, LA, USA. IEEE, June 2011. DOI: `10.1109/CEC.2011.5949963`. ISSN: 1941-0026.

[245] K. Li, M. Liao, K. Deb, G. Min and X. Yao. Does Preference Always Help? A Holistic Study on Preference-Based Evolutionary Multiobjective Optimization Using Reference Points. *IEEE Transactions on Evolutionary Computation*, 24(6):1078–1096, December 2020. ISSN: 1941-0026. DOI: 10.1109/TEVC.2020.2987559.

[246] J. Branke, K. Deb, K. Miettinen and R. Słowiński, editors. *Multiobjective Optimization: Interactive and Evolutionary Approaches*, volume 5252 of *Lecture Notes in Computer Science*. Springer, Berlin, Heidelberg, 2008. ISBN: 978-3-540-88907-6 978-3-540-88908-3. DOI: 10.1007/978-3-540-88908-3.

[247] L. Ben Said, S. Bechikh and K. Ghedira. The r-Dominance: A New Dominance Relation for Interactive Evolutionary Multicriteria Decision Making. *IEEE Transactions on Evolutionary Computation*, 14(5):801–818, October 2010. ISSN: 1941-0026. DOI: 10.1109/TEVC.2010.2041060.

[248] E. Filatovas, O. Kurasova and K. Sindhya. Synchronous R-NSGA-II: An Extended Preference-Based Evolutionary Algorithm for Multi-Objective Optimization. *Informatica*, 26(1):33–50, 2015. DOI: 10.15388/Informatica.2015.37.

[249] C. M. Fonseca and P. J. Fleming. Genetic Algorithms for Multiobjective Optimization: Formulation Discussion and Generalization. In *Proceedings of the 5th International Conference on Genetic Algorithms*, pages 416–423, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc., June 1993. ISBN: 978-1-55860-299-1.

[250] L. Thiele, K. Miettinen, P. J. Korhonen and J. Molina. A preference-based evolutionary algorithm for multi-objective optimization. *Evolutionary Computation*, 17(3):411–436, September 2009. ISSN: 1063-6560. DOI: 10.1162/evco.2009.17.3.411.

[251] J. Molina, L. V. Santana, A. G. Hernández-Díaz, C. A. Coello Coello and R. Caballero. G-dominance: Reference point based dominance for multiobjective metaheuristics. *European Journal of Operational Research*, 197(2):685–692, September 2009. ISSN: 0377-2217. DOI: 10.1016/j.ejor.2008.07.015.

[252] E. Filatovas, A. Lančinskas, O. Kurasova and J. Žilinskas. A preference-based multi-objective evolutionary algorithm R-NSGA-II with stochastic local search. *Central European Journal of Operations Research*, 25(4):859–878, 2017. DOI: 10.1007/s10100-016-0443-x.

[253] K. Deb and A. Kumar. Interactive evolutionary multi-objective optimization and decision-making using reference direction method. In *Proceedings of the 9th annual conference on Genetic and evolutionary computation*, GECCO '07, pages 781–788, New York, NY, USA. Association for Computing Machinery, July 2007. ISBN: 978-1-59593-697-4. DOI: 10.1145/1276958.1277116.

[254] K. Deb. Multi-objective Evolutionary Algorithms: Introducing Bias Among Pareto-optimal Solutions. In A. Ghosh and S. Tsutsui, editors, *Advances in Evolutionary Computing: Theory and Applications*, Natural Computing Series, pages 263–292. Springer, Berlin, Heidelberg, 2003. ISBN: 978-3-642-18965-4. DOI: 10.1007/978-3-642-18965-4_10.

[255] K. Deb and A. Kumar. Light beam search based multi-objective optimization using evolutionary algorithms. In *2007 IEEE Congress on Evolutionary Computation*, pages 2125–2132, Singapore. IEEE, September 2007. DOI: `10.1109/CEC.2007.4424735`. ISSN: 1941-0026.

[256] L. Li, F. Yao, N. Jing and M. Emmerich. Preference incorporation to solve multi-objective mission planning of agile earth observation satellites. In *2017 IEEE Congress on Evolutionary Computation (CEC)*, pages 1366–1373, San Sebastián, Spain. IEEE, June 2017. DOI: `10.1109/CEC.2017.7969463`.

[257] C. A. Coello Coello and M. Reyes Sierra. A Study of the Parallelization of a Coevolutionary Multi-objective Evolutionary Algorithm. In R. Monroy, G. Arroyo-Figueroa, L. E. Sucar and H. Sossa, editors, *MICAI 2004: Advances in Artificial Intelligence*, pages 688–697, Berlin, Heidelberg. Springer, 2004. ISBN: 978-3-540-24694-7. DOI: `10.1007/978-3-540-24694-7_71`.

[258] J. Blank and K. Deb. Pymoo: multi-objective optimization in python. *IEEE Access*, 8:89497–89509, 2020. DOI: `10.1109/ACCESS.2020.2990567`.

[259] K. Deb and R. B. Agrawal. Simulated binary crossover for continuous search space. *Complex Systems*, 9(2):115–148, 1995.

[260] K. Deb and M. Goyal. A combined genetic adaptive search (geneas) for engineering design. *Computer Science and Informatics*, 26(4):30–45, January 1996. ISSN: 0254-7813.

[261] K. Deb, L. Thiele, M. Laumanns and E. Zitzler. Scalable test problems for evolutionary multi-objective optimization. In volume 112, pages 1–27. Eidgenössische Technische Hochschule Zürich (ETH), Institut für Technische Informatik und Kommunikationsnetze (TIK), 2001. DOI: `10.3929/ethz-a-004284199`.

[262] K. Deb and D. Saxena. Searching for pareto-optimal solutions through dimensionality reduction for certain large-dimensional multi-objective optimization problems. In *Proceedings of the World Congress on Computational Intelligence (WCCI-2006)*, pages 3352–3360, Vancouver, Canada. IEEE, January 2006.