Design Space Exploration of Approximate Computing Techniques

Design Space Exploration of Approximate Computing Techniques

Design Space Exploration of Approximate Computing Techniques

SEPIDE SAEEDI, ALESSANDRO SAVINO, and STEFANO DI CARLO, Politecnico di Torino, Italy

Nowadays, the rising energy consumption of smartphones and portable devices creates an energy efficiency challenge. To address this problem, Approximate Computing (AxC) techniques are becoming popular since they sacrifice computation accuracy for enhanced performance, energy efficiency, and area reduction. However, selecting suitable AxC techniques for target applications remains intricate. Design Space Exploration (DSE) approaches can be employed to systematically explore all different possible approximate versions of an application and select the most suitable versions. This paper proposes a DSE approach that models the target application computations and the approximation-induced errors using Interval Arithmetic. The experimental results show the efficiency of the proposed approach in quickly evaluating different approximate versions of an application eliminating the time-consuming task of executing each approximate version. Also, using Artificial intelligence, such as Reinforcement Learning approaches, is proposed to explore the design space automatically.

CCS Concepts: • **Hardware → Power estimation and optimization**; **Electronic design automation**; • **Computing methodologies → Representation of mathematical functions**.

Additional Key Words and Phrases: Approximate Computing, Design Space Exploration, Interval Arithmetic, Reinforcement Learning.

ACM Reference Format:

Sepide Saeedi, Alessandro Savino, and Stefano Di Carlo. 2023. Design Space Exploration of Approximate Computing Techniques. 1, 1 (August 2023), 7 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

The overall energy consumption of computers and communication devices is increasing rapidly, and soon modern computer devices will require more energy than energy resources can provide [2]. The increasing use of technologies such as smartphones, smart devices, sensors, and the Internet of Things has become deeply ingrained in our lives, leading to an increased need for computing power and communication capabilities [9]. Energy efficiency can be achieved by optimizing both power consumption and computation time. The reason is that the energy consumption of computers and communication devices can be defined as the product of time and average power consumption per operation [2]. However, achieving fast computations with limited energy resources is non-trivial. Reducing power consumption may increase computation time, while achieving faster computation may require higher power consumption.

Approximate computing (AxC) offers an effective solution to address this challenge by shifting the trade-off paradigm. Rather than solely trading off power consumption and computation time, AxC techniques introduce the concept of trading off computation accuracy for improvements in power consumption and computation time [12]. AxC allows inexact computations, acknowledging that specific applications can tolerate a certain degree of accuracy loss without significantly impacting their overall functionality [3]. In other words, by intentionally relaxing precision requirements,

Authors' address: Sepide Saeedi, sepide.saeedi@polito.it; Alessandro Savino, alessandro.savino@polito.it; Stefano Di Carlo, stefano.dicarlo@polito.it, Politecnico di Torino, Turin, Italy.

Manuscript submitted to ACM

AxC provides an avenue to achieve gains in power consumption and computation time while maintaining an acceptable level of accuracy for a given application.

Reducing accuracy through AxC techniques is acceptable in scenarios where the introduced inaccuracies have a tolerable impact on the overall quality of results, but the benefits in terms of reduced power consumption or computation time are significant. This trade-off between accuracy and efficiency is grounded in the observation that many applications do not require absolute precision and can still function well with slightly imprecise or approximate results. For example, in image and video compression algorithms, a slight loss of image quality may be imperceptible to the human eye. The data representation can be simplified using AxC techniques like quantization, where color values are rounded to the nearest value in a smaller set. This approximation reduces the amount of data stored or transmitted, leading to lower power consumption and faster data transfer. Also, in some machine learning models, minor inaccuracies in intermediate calculations may not significantly affect the final model's accuracy. AxC techniques like reduced-precision training or quantization allow storing and computing with lower precision weights and activations. This results in faster inference times and reduced memory usage, leading to energy efficiency gains.

To better understand how AxC techniques might be applied to an application, the computation flow of an application is depicted in figure 1. The application receives three inputs a, b, and c. Multiplies input a by a constant w, then adds input c to the product and generates output O_1. In parallel, input c is subtracted from input a, producing output O_2 while inputs b and c are added together and produce output O_3. Here, input b, constant w, and the subtraction are chosen to apply AxC techniques. A round to nearest AxC technique is applied to b and w, and the exact subtractor is replaced with an approximate one. In this approximate version of the application, all outputs are impacted by approximation since in the path from inputs to each output, at least one AxC technique was applied, leading to the accuracy loss at outputs. Any other AxC techniques applied to any selection of operands and operators can produce a different approximate version of the application with different levels of accuracy loss at outputs.

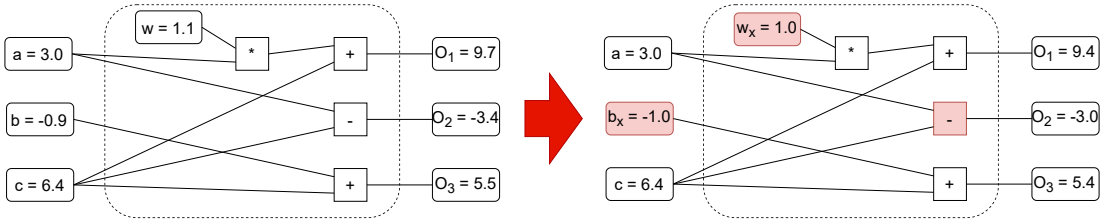

Fig. 1. examples of computation flow: a) exact version and b) an approximate version.

For most applications, the computation flow is more complex than the example in Figure 1. In most cases, evaluating all possible approximate versions of an application and selecting the most suitable ones in terms of reducing power consumption and computation time is quite challenging and even may become infeasible in a reasonable time. This problem can be viewed as a Design Space Exploration (DSE) problem. DSE can be defined as the systematic exploration of the possible design choices and configurations of a system or software application. In other words, DSE is a process to identify and evaluate different design alternatives to find an optimal or satisfactory solution based on specific criteria or objectives. In this concept, the design space is composed of different approximate versions of the application, and the problem is to systematically evaluate and compare all approximate versions based on objectives such as power consumption and computation time reductions.

Different approaches have been proposed in the literature to efficiently explore a complex design space with the aim of reducing the number of designs and the exploration time. [5, 6, 10, 13]. Two different DSE approaches are proposed in this work, to explore a complex design space of an application's approximate versions, more efficiently. The first proposed approach explained in subsection 2.1, aims to reduce the exploration time while performing the DSE on all approximate versions of the application and provide an acceptable estimation of the accuracy loss at outputs for each approximate version. The DSE is performed by modeling the application computation flow and resorting to Interval Arithmetic (IA) [1] concepts to model the approximation-induced errors. However, in some cases, exploring all possible approximate versions of the application that comprise the design space in a reasonable time is infeasible. Hence, another approach is proposed explained in subsection 2.2, that aims to perform the DSE in a reasonable time efficiently and intelligently, omitting the need to explore the whole design space. The DSE is performed by leveraging an Artificial intelligence concept called Reinforcement Learning (RL).

2 METHODOLOGY

In this section, the proposed approaches are presented. The proposed DSE approach using IA concepts is explained in subsection 2.1, and the proposed DSE approach using RL is described in subsection 2.2.

2.1 Interval Arithmetic

Considering the computation flow of an application, when a mathematical operation or its operands are approximated, the approximation-induced error propagates into the computation flow until it reaches the application output. An example of this is depicted in figure 1. Once the approximation-induced errors at the outputs for each approximate version of the application are calculated, the approximate versions can be compared against each other regarding the accuracy loss they impose on the application outputs. Then, it can be decided which AxC technique is the most suitable for the target application based on evaluating the consequent output accuracy degradation for each AxC technique against the gains in reducing power consumption, computation time, or memory size. Hence, modeling the computation flow of the target application is proposed. Then the approximation error was modeled using the IA paradigm.

IA differs from the arithmetic mathematical approach usually used. In arithmetic, mathematical operations are performed on specific numbers, resulting in a single, precise outcome for each calculation. For example, 2 + 3 equals 5 without considering uncertainties or intervals. On the other hand, IA works with ranges of numbers known as intervals. Each operation in IA yields a range of outcomes. For instance, when adding the intervals [2, 3] and [3, 5], the result is [5, 8], meaning that any number obtained by adding elements from the first interval to elements from the second interval will lie within the range of 5 to 8. IA is particularly useful when dealing with imprecise data, uncertainties, or rounding errors in numerical computations.

The AxC technique used is a precision reduction performed on the target application parameters. Precision reduction is an AxC technique that reduces the precision of each number. For example, if the number is demonstrated as a 32-bit fixed-point number with 16 bits for the fractional part, as shown in the figure 2, a 1-bit precision reduction means that 1 bit is removed from the end of the fixed-point number.

To quickly evaluate the impact of different precision reductions on the accuracy degradation of the target application output, the approximation-induced error ranges are demonstrated as intervals using the IA concepts. Also, all the operands of mathematical operations in the computation flow are demonstrated as intervals. Then, all mathematical operations in the application computation flow can be performed using IA concepts. Now, every number presenting the value of each operand in the computation flow is demonstrated as $v - [\epsilon]$. For example, in figure 1, the value of

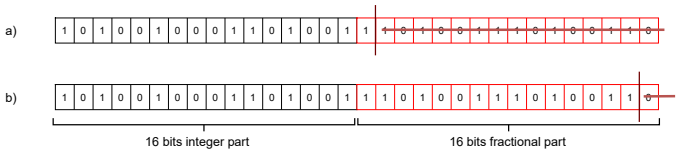

Fig. 2. examples of precision reduction on a 32-bit fixed point number: a) 15-bit precision reduction and b) 1-bit precision reduction.

parameter w is 1.1 before approximation. Hence it can be demonstrated as an IA number with $1.1 - [0, 0]$. After applying the approximation, the value of parameter w changes to 1.0, so the IA number demonstrating the approximated value as the difference between the exact value and the approximation error is $1.1 - [0.1, 0.1]$.

In cases where the error value is not a range and can be presented with a specific number, the intervals might not seem helpful. However, the benefit of this presentation is that it can demonstrate and calculate the approximation errors separate from the exact values. Hence, the same model can be used by changing the error interval to calculate the impact of another AxC technique that may impose a range of errors like $[0, 0.5]$. Alternatively, when calculating the impact of several AxC techniques or different AxC configurations, the error intervals can be accumulated to show the possible errors imposed by applying all the AxC techniques on the same operand. As a result, the approximation error on the application output can be calculated with one model run for many approximate versions and, in the end, evaluate the error interval imposed by all the precision reduction configurations.

2.2 Reinforcement Learning

When the design space is so vast that exploring all possible approximate versions in a reasonable time is infeasible, the approach presented in subsection 2.1 is not helpful. So, using RL concepts is proposed to perform the DSE in a reasonable time efficiently and intelligently, omitting the need to explore the whole design space. To address complex DSE, machine learning approaches such as RL are proposed in the literature [5, 13].

In RL, an "agent" learns optimal behavior by interacting with a changing "environment" and receiving feedback in the form of "observations" and "rewards." This learning process involves repeated trial and error [7]. The agent and environment interactions are depicted in figure 3. Our approach involves using RL to automate the DSE. The aim is to find favorable trade-offs between accuracy degradation and power/computation time. The agent is the learning algorithm that interacts with the environment. The environment is the system in which the agent operates. It models all the optimization aspects, including the approximate mathematical operators. The RL agent interacts with the environment by applying an action to the environment, which includes information about which approximate operators the agent has selected. Then the environment's current state is set to the selection of approximate operators that the agent just requested. The environment generates an approximate version of the application based on the current agent's action. The agent then receives observations about accuracy, power, and computation time for the current approximate configuration and a reward reflecting the approximate configuration's desirability. The agent learns to make decisions for its next action, striving to maximize cumulative rewards by mapping environment states to actions. While this approach is implemented, additional experimentation is required to comprehensively validate its effectiveness.

3 EXPERIMENTAL SETUP AND RESULTS

In this section, an overview of our experimental setup and results for the DSE proposed IA-based approach is provided, followed by a brief explanation of the work in progress and the preliminary results.

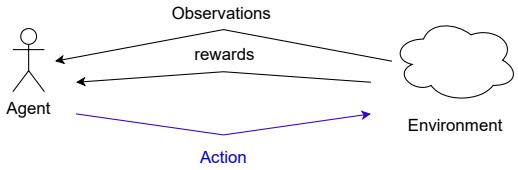

Fig. 3. Reinforcement Learning Agent interacts with the Environment.

A Spiking Neural Network (SNN) was selected as a target application. SNNs are Artificial Neural Networks (ANNs) inspired by the brain's neuron communication via electrical spikes [8]. SNNs, like other ANNs, simulate biological neural networks for tasks such as pattern recognition and image classification. After training, ANNs develop learned patterns within their interconnected nodes called neurons and weighted connections. During the test phase, input data is fed into the network, propagating through the layers and activating specific nodes based on learned features. The final output provides the model's prediction or classification result, enabling it to recognize patterns or objects within the input data. A trained SNN from [4] with a layer of 400 identical neurons was selected as a target application. The inputs are the images of handwritten digits from the MNIST data set [14] translated into data sequences, while the SNN primary outputs are the spike counts obtained at the end of each neuron computation flow. The SNN classification output is obtained by voting among the primary outputs (spike counts) to indicate which digit is most probably the digit in the handwritten image.

The original SNN was first trained in full precision (floating-point), then all parameters were converted to 32-bit fixed-point values. The precision reduction was performed on the 16-bit fractional part. Hence, 15 different precision reductions are possible since 1 to 15 bits can be cut from the rightmost side of a number's fractional part.

To explore the design space, the application must be executed 15 times, each time for a different precision reduction applied, to evaluate the impact of each precision reduction on the output accuracy separately. Instead, the IA-based model of the application computation flow can quickly calculate the outputs and their accuracy degradation, only running the application 3 times, each time for evaluating 5 precision reductions. Knowing the error interval imposed by each precision reduction, the overall error interval for each group of 5 precision reductions can be calculated based on the IA concepts. The first group includes 1- to 5-bit precision reductions, the second group includes 6- to 10-bit precision reductions, and the third group includes 11- to 15-bit precision reductions.

The results for a subset of 30 MNIST test set images were collected. Suppose the estimation error is the difference between the output spike count obtained by the IA-based model and the output spike count obtained by executing the original approximated networks. This error can show how closely the IA-based model can follow the original model in producing the outputs. The minimum, maximum, and average errors among the three groups were 0, 10, and 3.2167. This error is acceptable since the SNN classification results obtained with the IA-based model are identical to those from the original SNNs. It is important to note that the time required to run the evaluation is one-fifth before, while the network classification accuracy is untouched. For further details, the reader may refer to [11].

To further investigate the opportunities of this approach, the approximation-induced error is observed at specific points of the computation flow by monitoring the output of selected arithmetic operations while using the IA-based model to evaluate the impact of all precision reductions at once. The exploration starts with a 15-bit precision reduction of all the network parameters. At the end of the run for the first image, each primary output (spike counter) is checked. Since each neuron has one spike count final output, this means that the output of each neuron is observed.

If the approximation-induced error interval at this point does not exceed the acceptable error range assigned to this observation point, the 15-bit precision reduction is acceptable for all the network parameters involved in calculating this specific spike counter. Otherwise, all the observation points before this specific spike counter should be checked from the output side (the spike counter) to the input side. Then, one bit is restored to the fractional part for all the parameters before the observation point that shows a violation in their acceptable error range. This procedure continues until no observation point shows a violation in its acceptable error range.

Using this approach for the same SNN run in the test phase with 1000 images of the MNIST test set, the preliminary results show that, on average, for each image, the exploration stops after 8 rounds of exploration while the SNN classification accuracy is untouched. The results for an example image in figure Figure 4 show that in the first round of exploration with 15-bit precision reduction, acceptable error range violation is observed at all primary outputs, which means that 15-bit precision reduction for all SNN parameters is not suitable. The same happens until the 5th exploration round when 91.75% of the output observations show that more precision is needed for the parameters before that point, while 8.25% of the output observations show that 10-bit precision reduction is acceptable for the parameters before that point. Finally, at the 8th round of exploration, the remaining network parameters accept a 7-bit precision reduction.

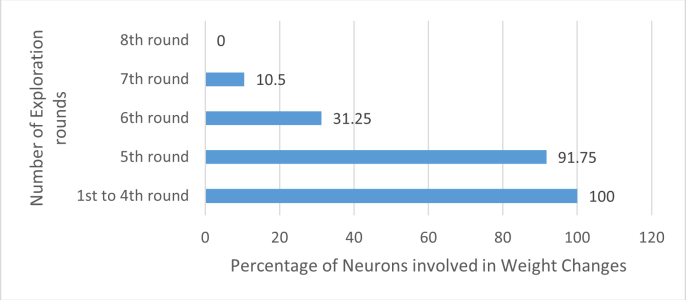

Fig. 4. Percentage of neurons involved in changing SNN parameters for one image

With this approach, the minimum fractional bit width for each network parameter is indicated separately, in a reasonable exploration time. Notably, this approach covers a more extensive design space since the same precision reduction is not applied to all network parameters.

4 CONCLUSION

We proposed approaches to reduce the exploration time needed to perform a Design Space Exploration (DSE) for selecting the most suitable approximate version of an application. The Interval Arithmetic based approach proved effective in estimating the impact of precision reduction on the application accuracy without executing each approximate version, reducing the DSE time significantly. However, when DSE in an acceptable time is infeasible, using a Reinforcement Learning approach is proposed to intelligently and efficiently explore the space. Further experiments are conducted to validate the effectiveness of this approach.

ACKNOWLEDGMENTS

This work has received funding from the APROPOS project in the European Union's Horizon 2020 research and innovation program under the Marie Skłodowska-Curie grant agreement No 956090.

REFERENCES

[1] 2015. IEEE Standard for Interval Arithmetic. *IEEE Std 1788-2015* (2015), 1–97. https://doi.org/10.1109/IEEESTD.2015.7140721

[2] Alberto Bosio, Stefano Di Carlo, Patrick Girard, Ernesto Sanchez, Alessandro Savino, Lukas Sekanina, Marcello Traiola, Zdenek Vasicek, and Arnaud Virazel. 2020. Design, verification, test and in-field implications of approximate computing systems. In *2020 IEEE European Test Symposium (ETS)*. IEEE, 1–10.

[3] Alberto Bosio, Daniel Ménard, and Olivier Sentieys (Eds.). 2022. *Approximate Computing Techniques*. Springer International Publishing. https://doi.org/10.1007/978-3-030-94705-7

[4] Alessio Carpegna, Alessandro Savino, and Stefano Di Carlo. 2022. Spiker: FPGA-Optimized Hardware Accelerator for Spiking Neural Networks. In *2022 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*. 1–6.

[5] Quentin Gautier, Alric Althoff, Christopher L Crutchfield, and Ryan Kastner. 2022. Sherlock: A multi-objective design space exploration framework. *ACM Transactions on Design Automation of Electronic Systems (TODAES)* 27, 4 (2022), 1–20.

[6] Wei Hu, Jianlei Zhao, Yuchun Yang, and Ying Wang. 2019. Exploring the design space of approximate arithmetic circuits using reinforcement learning. In *2019 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 442–447.

[7] Leslie Pack Kaelbling, Michael L Littman, and Andrew W Moore. 1996. Reinforcement learning: A survey. *Journal of artificial intelligence research* 4 (1996), 237–285.

[8] Wolfgang Maass. 1997. Networks of spiking neurons: The third generation of neural network models. *Neural networks* 10, 9 (1997), 1659–1671.

[9] Aleksandr Ometov and Jari Nurmi. 2022. Towards approximate computing for achieving energy vs. accuracy trade-offs. In *2022 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 632–635.

[10] Michalis Rizakis, Stylianos I. Venieris, Alexandros Kouris, and Christos-Savvas Bouganis. 2018. Approximate FPGA-Based LSTMs Under Computation Time Constraints. In *Applied Reconfigurable Computing. Architectures, Tools, and Applications*, Nikolaos Voros, Michael Huebner, Georgios Keramidas, Diana Goehringer, Christos Antonopoulos, and Pedro C. Diniz (Eds.). Springer International Publishing, Cham, 3–15.

[11] Sepide Saeedi, Alessio Carpegna, Alessandro Savino, and Stefano Di Carlo. 2022. Prediction of the Impact of Approximate Computing on Spiking Neural Networks via Interval Arithmetic. In *2022 IEEE Latin-American Test Symposium (LATS)*.

[12] Phillip Stanley-Marbell, Armin Alaghi, Michael Carbin, Eva Darulova, Lara Dolecek, Andreas Gerstlauer, Ghayoor Gillani, Djordje Jevdjic, Thierry Moreau, Mattia Cacciotti, Alexandros Daglis, Natalie Enright Jerger, Babak Falsafi, Sasa Misailovic, Adrian Sampson, and Damien Zufferey. 2020. Exploiting Errors for Efficiency: A Survey from Circuits to Applications. *ACM Comput. Surv.* 53, 3, Article 51 (jun 2020), 39 pages. https://doi.org/10.1145/3394898

[13] Yue Wu, Peng He, Yuhang Li, Zhigang Wang, Hao Zhang, and Zhonghai Zhu. 2021. Ironman: Reinforcement learning based design space exploration for approximate computing. In *2021 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. IEEE, 1–8.

[14] Corinna Cortes Yann LeCunn and Christopher J.C Burges. [n. d.]. *The MNIST database*. http://yann.lecun.com/exdb/mnist/