Design Space Exploration Frameworks for Approximate Computing

Olivier Sentieys[1], Daniel Ménard[2], Alberto Bosio[3]
[1]University of Rennes, INRIA/IRISA - France - Email: olivier.sentieys@inria.fr
[2]INSA Rennes - IETR - France - Email: daniel.menard@insa-rennes.fr
[3]École Centrale de Lyon - INL - France - Email: alberto.bosio@ec-lyon.fr

Approximate Computing (AxC) term indicates a design paradigm that aims to selectively violate the specifications, trading accuracy off for efficiency. For some kinds of applications it has been demonstrated in the literature the effectiveness of imprecise computation for both software and hardware components implementing inexact algorithms, showing an inherent resiliency to errors [1]. Despite many works explored the possibilities given by AxC paradigm, there are still open challenges that hold back AxC from a wider employment. In particular, the key point is a lack of a general automation tool and a methodology for the design space exploration that searches for the best trade-off between the grade of the inaccuracy of an approximate algorithm and the gain in efficiency.

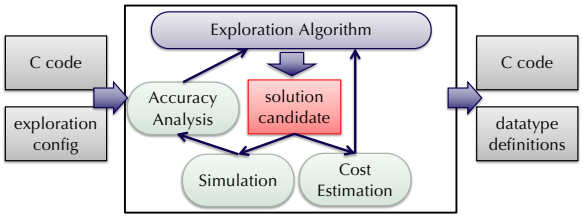

Fig. 1: Design Space Exploration Flow

In this work, we present a design exploration tool for approximate algorithm, which is an automatic framework able to find approximate versions for an application described as C/C++, by mutating the original code. Figure 1 depicts the entire flow. It consists of two main components: (i) a source-to-source manipulation tool, which analyzes the C source code to apply user-defined code mutators, and (ii) an exploration algorithm search engine, which aims to find the best functional approximation version of a given C code, with respect to user-defined objective functions.

The first demonstration relates to the word-length optimization of variables in a software or hardware system to explore cost (e.g., energy) and quality trade-off solution. The tool is scalable and targets both customized fixed-point and floating-point arithmetic. Fixed-point arithmetic is widely used for the implementation of various applications, from signal and image processing to machine learning, on embedded systems [2]. Customized floating-point is also an emerging paradigm for energy-efficient accelerators, especially in machine learning applications [2]. Designing such accelerators always require to optimize the word-length of variables and operations in the system to find a trade-off between cost, performance, energy and quality requirements. This process of Word-Length Optimization (WLO), known as an NP-hard problem with the complexity growing exponentially when more variables are involved, may take up to 25 - 50% of design time [3]. Therefore, WLO is still a key problem during the design of software or hardware systems. A common approach to WLO involves iterative search algorithms based on simulations or relaxing the problem with convex analytical functions to reduce optimization time [4]. Recently, some techniques to address the scalability of WLO have been proposed. Some approaches first perform local WLO with different accuracy constraints and later combine the local solutions with an iterative search. Other techniques have explored hierarchical decomposition of the original system to reduce the problem size at each step in the exploration. In this demonstration, we discuss an approach to overcome scalability issues when applying WLO for applications with several kernels. We use Image Signal Processor (ISP), a post-processing pipeline for digital cameras, as our running example. The ISP consists of several kernels such as noise cancelling (Non-Local Means), linearization recovering, light- fall-off correcting (Vignetting Correction), color adjusting and quality improving by the remaining kernels.

The second demonstration is 𝕀DE𝔸 (𝕀DE𝔸 Is a Design Exploration tool for Approximate Algorithm), which is an automatic framework able to find approximate versions for an application described as C/C++, by mutating the original code. 𝕀DE𝔸 does not need to specify which part(s) of the application should be approximated and how. It only requires the definition of the acceptable output degradation from the user. The output can be either a software or hardware implementation of the approximated application, accordingly to involved approximate technique(s). 𝕀DE𝔸 consists of two main components: (i) a source-to-source manipulation tool, the clang-Chimera [5], which analyzes the Abstract Syntax Tree (AST) of a C/C++ code to apply user-defined code mutators, and (ii) an Evolutionary search engine, the Bellerophon, which aims to find the best functional approximation version of a given C/C++ code, with respect to user-defined objective functions

References

[1] S. Mittal, "A survey of techniques for approximate computing," *ACM Comput. Surv.*, vol. 48, no. 4, pp. 62:1–62:33, Mar. 2016. [Online]. Available: http://doi.acm.org/10.1145/2893356
[2] B. Barrois and O. Sentieys, "Customizing Fixed-Point and Floating-Point Arithmetic - A Case Study in K-Means Clustering," in *IEEE International Workshop on Signal Processing Systems (SiPS)*, Oct. 2017.
[3] K. Parashar, D. Menard, and O. Sentieys, "A polynomial time algorithm for solving the word-length optimization problem," in *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, 2013.
[4] H.-N. Nguyen, D. Ménard, and O. Sentieys, "Novel Algorithms for Word-length Optimization," in *Proc. 19th European Signal Processing Conference (EUSIPCO)*, 2011, pp. 1944–1948.
[5] M. Barbareschi, F. Iannucci, and A. Mazzeo, "A pruning technique for b amp;amp;b based design exploration of approximate computing variants," in *2016 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, July 2016, pp. 707–712.