Efficient Design Space Exploration of OpenCL Kernels for FPGA Targets Using Black Box Optimization

ALIREZA GHAFFARI, AND YVON SAVARIA, (Fellow, IEEE)
Department of Electrical Engineering, Polytechnique Montréal, Montréal, QC H3T 1J4, Canada
Corresponding author: Alireza Ghaffari (seyed-alireza.ghaffari@polymtl.ca)

ABSTRACT Nowadays, many industries are in favor of using intelligent design space exploration as opposed to brute-force analysis. In many applications, the design space is defined by multiple variables and their interactions. Although brute-force analysis is very simple, it is rarely scalable when the number of variables in the system increases. With the rising complexity of hardware designs, more intelligent approaches are needed to explore the design options. This paper proposes using smart meta-heuristic search algorithms such as Grey Wolf Optimization (GWO) in conjunction with Bayesian Optimization (BO) to solve this problem. We show that we can further reduce the design effort using a surrogate model that is created based on a novel hybrid GWO-BO method. The surrogate model is a useful abstraction to detect functional and physical inter-dependencies in the system in order to accurately predict its performance (e.g. throughput or latency). We evaluate our methodology and show that it can produce competitive results in order to find the best design variables that maximize the performance of the system. Finally, we compare our results with previous statistical and heuristic methods proposed in the literature and find that the proposed GWO-BO method always performs better than the other considered methods.

INDEX TERMS Accelerator design, Bayesian optimization, design space exploration, grey wolf optimization, hardware design, surrogate model.

I. INTRODUCTION
Designing efficient hardware is a ubiquitous requirement in industrial and scientific problems. To achieve the best efficiency, designers need to take into account the effect of many design variables. Exploring all possible combinations of variables is often impractical. On the other hand, the interaction between variables and their related trade-offs are often unknown or differ from one design to another. More specifically, in hardware design, synthesizing a circuit to evaluate one set of variables can take hours, while designers must explore tens of thousands of design options to find a high quality solution. Moreover, since there is no gradient defined for most hardware variables (e.g. memory block size, number of parallel data paths, etc), gradient-based optimization is not a viable option to explore the design space.

In such a complex multi-dimensional optimization problem, human expert choices can be inaccurate or biased. Furthermore, understanding and estimating a multi-dimensional, non-linear non-convex objective function is difficult for most human experts, and it is easy to miss optimal design solutions.

In particular, with the rise of High-Level Synthesis (HLS), it is possible to design hardware solutions using high-level languages such as C/C++ and OpenCL. While HLS makes the design process easier and faster, understanding the design space is harder. For instance, when using OpenCL, the increasing number of threads and number of SIMD (Single Instruction Multiple Data) instructions per thread both seem to increase performance. However, finding the optimum choices for threads and SIMD elements per thread to achieve the best throughput/area usage is not obvious.

It is commonly known that using meta-heuristics such as Genetic Algorithm (GA) [1] or derivative-free optimizations [2] can help with such design optimization problems. As we will explore in this paper, the issue with meta-heuristic algorithms is that they do not provide any understanding or estimation of the objective function. On the other hand, it is also well known that statistical models such as Bayesian Optimization (BO) [3] and LASSO (least absolute shrinkage and selection operator) [4] can provide some understanding.
of the model behavior. However, statistical models such as BO are extremely sensitive to the data samples provided to them. This means that if the samples are not correctly drawn from the environment, the surrogate model might not behave like the actual model.

In this paper, we propose algorithms that perform design space exploration for OpenCL kernels. By using a simple Hill-Climbing algorithm, we show that a hill climber often reaches a local maximum (or minimum), thereby demonstrating that the design space is often non-convex. Next, we propose Grey Wolf Optimization (GWO) as a meta-heuristic to explore the design space. Then, Bayesian Optimization (BO) with random samples from the environment is proposed to create a surrogate model of the system. Finally, by creating a hybrid optimizer (GWO-BO), we show that the quality of the samples explored by the wolves in the GWO algorithm is better than random samples in order to obtain a surrogate model that matches the maximum performance achievable by the actual physical design.

This work makes the following contributions:
- Providing an automated framework to optimize OpenCL kernels in HLS.
- Proposing that Grey Wolf Optimization (GWO) is a competitive method to perform design space exploration.
- Building surrogate models that estimate the behavior of OpenCL kernels using a Bayesian Optimizer (BO).
- Proposing a hybrid (GWO-BO) optimizer based on re-using the points explored by the GWO algorithm and feeding them to BO to create a surrogate model for HLS OpenCL kernels that systematically provide results better than both GWO and BO methods.

To the best of our knowledge 1) this is the first time that GWO is proposed for hardware design space exploration and 2) this is the first time that the explored variables of a meta-heuristic algorithm are re-used to build a more accurate surrogate model (i.e. Hybrid GWO-BO method). The hybrid method enables benefiting from both algorithms without repeating expensive synthesis trials.

The rest of this paper is organized as follows: In Section.II, we review the literature related to this topic. In Section.III, we introduce three design space exploration methods as well as some background information regarding the algorithms and benchmarks that are used in this paper. In Section.IV, we show the results from each algorithm and compare them with the benchmark. In Section.V, we study multi-objective optimization of throughput and area utilization, as well as Pareto-efficiency of the design variables. Finally, we compare our results with those obtained with previously proposed methods.

II. RELATED WORKS

Recently, intelligent design space exploration has attracted much attention among hardware designers and scientists. Hardware design space exploration can be treated as a black-box optimization problem because first, evaluating the objective function is expensive, and second, the properties of the objective function such as derivatives and convexity are unknown. For years, brute-force analysis for small design spaces and meta-heuristic algorithms for larger design spaces were commonly used by designers to perform the exploration. For instance, meta-heuristic optimization algorithms such as the Genetic Algorithm [5] or Simulated Annealing [6] are typically used to explore large design spaces. The main problem with meta-heuristic methods is that, although they are very effective in finding optimum solutions, they cannot provide a suitable overall estimation of the system behavior (i.e. system’s surrogate model). Abstract estimation of a system’s behavior is immensely important when trying to understand the inter-dependencies of the design variables.

On the other hand, BO [7] is an interesting approach for design space exploration because it is a form of incremental learning that provides an estimation of the objective function based on the previously observed samples. This is a very powerful strategy for finding the extremum of the objective functions that are not easy to evaluate. Furthermore, BO can estimate a surrogate model of the system that helps to explain the inter-dependencies of design variables. For instance, in [8], the authors suggested a framework for design space exploration for C/C++ High-Level Synthesis that uses BO. They have shown that BO outperforms the traditional search method in terms of latency and resource usage in the FPGA targets. Similarly, in [3], the authors used BO to optimize hardware accelerators for deep neural networks. They reported that the BO method could help to simulate the accuracy of the deep neural network and energy efficiency of the accelerator hardware. In [9], BO was applied to tune directives to achieve minimum latency.

Although the approach we are presenting also uses BO to perform design space exploration, there are two distinct features that differentiate our work from [3], [8]. First, our work is based on High-Level Synthesis for OpenCL kernels. Design space exploration for OpenCL kernel poses new challenges as the fine-grained thread-level parallelism must be controlled for the FPGA device. Moreover, as opposed to the above mentioned work, we did not discard the meta-heuristic methods. Our work also explores using a new meta-heuristic method, Grey Wolf Optimization [10], that is suitable for design space exploration. Furthermore, we combined GWO and BO methods to obtain the best results.

Inspired by traditional design space exploration using meta-heuristics, we believe there is still room for more research using meta-heuristic methods and their combination with BO. For instance, the recently suggested Grey Wolf Optimization (GWO) [10] exhibits competitive performance in design space exploration applications. In this research, we show that samples that have been chosen by the GWO algorithm can be very suitable candidates for constructing a surrogate model that estimates the minimum latency.

Reinforcement learning [11] has also been shown effective for design space exploration. Similar to Bayesian optimization, reinforcement learning is also an incremental learning.
method that directs the search agent toward the optimum solution by analyzing the previously observed samples using a Q-table or neural network. In [12], the authors used reinforcement learning to explore the optimization of deep neural networks on the ARM-Cortex-A CPUs. Likewise, in [13], the authors used a time-limited reinforcement learning [14] to execute design space exploration for deeply pipelined OpenCL kernels of the convolutional neural networks.

Finally, there are other machine learning-based exploration techniques, such as random forest [15] for HLS design space exploration, and Markov decision process [16] for exploration of multi-processor platforms that have appeared in the literature. Among those researches, Bayesian and meta-heuristic methods still exhibit more competitive exploration performance for large design spaces.

III. METHODOLOGY

The design space exploration problem can be considered as a Black-Box Optimization (BBO) [2] problem. Typically, a black-box optimizer is a type of optimizer that assumes the objective function is unknown. Furthermore, there is no assumption of any form of continuity, convexity, and smoothness of the objective function. The black-box methods that are used in this work are derivative-free, which is beneficial because, for most hardware design variables, the derivative does not exist or is hard to compute. Historically, meta-heuristic methods such as genetic algorithm [17] are well-known methods to solve the black-box optimization problem. Although meta-heuristics are fast at finding the optimum solution, they do not provide any abstract modeling (i.e. surrogate model) of the black box objective function. Moreover, they are not guaranteed to converge.

Fig. 1 summarizes the framework that is discussed in this paper. A design space exploration framework can be defined as a black box optimizer that receives resource utilization and throughput performance from either a physical design or a synthesizer tool. In this scheme, the design space explorer may save the samples that it receives from the environment in order to create a surrogate model of the objective space. Furthermore, the black box optimizer provides the optimum design variables to the OpenCL kernels.

In the following section, first, the hill-climbing method and its inefficiency for non-convex design spaces is reviewed. Second, Grey Wolf Optimization is proposed as a heuristic method for design space exploration and third, Bayesian optimization is reviewed as a tool to obtain the surrogate model of the objective function. Fourth, an ensemble method combining GWO and BO is proposed. Finally, after presenting the OpenCL benchmark suite that we used to evaluate our results, it is shown that the proposed GWO-BO method achieves the best results on that benchmark suite.

A. HILL-CLIMBING AS A MEASURE OF NON-CONVEXITY

Hill-Climbing [1] is an iterative numerical algorithm that starts from an arbitrary location in the design space and attempts to find a better solution by moving incrementally toward the optimum solution. This incremental process continues until there is no further improvement in the objective function. Although the Hill-Climbing algorithm is very efficient at finding local optimums, it usually fails to find the global optimum in a non-convex environment. Given that hill climbing is not a good strategy for non-convex design spaces, we use it in this paper as a measure of the non-convexity of a design space. Note that a hill climber might fail to find the global optimum as it gets stuck in a local optimum, this means the design space is non-convex.

As shown in Algorithm 1, a Hill-Climber algorithm starts from an initial variable choice $H_0 = (x_{10}, x_{20} \ldots x_{n0})$, which is chosen randomly in the design space. In every iteration, the optimizer visits all the neighbors of the current variable in order to create a surrogate model of the objective space. Furthermore, the black box optimizer provides the optimum design variables to the OpenCL kernels.

In the following section, first, the hill-climbing method and its inefficiency for non-convex design spaces is reviewed. Second, Grey Wolf Optimization is proposed as a heuristic method for design space exploration and third, Bayesian optimization is reviewed as a tool to obtain the surrogate model of the objective function. Fourth, an ensemble method combining GWO and BO is proposed. Finally, after presenting the OpenCL benchmark suite that we used to evaluate our results, it is shown that the proposed GWO-BO method achieves the best results on that benchmark suite.
In nature, grey wolves encircle their prey. This phenomenon can be mathematically formulated as:

\[
\tilde{D} = |\tilde{C} \tilde{X}_p(t) - \tilde{X}(t)|
\]

\[
\tilde{X}(t + 1) = \tilde{X}_p(t) - \tilde{A} \tilde{D}
\]  

where \(\tilde{D}\) is the relative distances of the wolves to the prey, \(\tilde{X}_p\) is the position of the prey, \(\tilde{X}(t)\) is the position of wolves on iteration \(t\), and \(\tilde{A}\) and \(\tilde{C}\) are coefficient vectors calculated by:

\[
\tilde{A} = 2\tilde{a} \tilde{r}_1 - \tilde{a}
\]

\[
\tilde{C} = 2 \tilde{r}_2
\]

where \(\tilde{r}_1\) and \(\tilde{r}_2\) are uniformly distributed random variables between \([0, 1]\) and elements of the vector \(\tilde{a}\) are linearly decreased from \(2\) to \(0\) as the algorithm iterates. Note that \(\tilde{a}\) controls the exploration and exploitation of the algorithm. When \(\tilde{a}\) is \(2\), the GWO is in the exploration mode and as it goes toward \(0\), the algorithm enters the exploitation mode.

Inspired by the hierarchy of wolves, the search is guided by the three fittest search agents namely \(\alpha\), \(\beta\) and \(\gamma\) wolves. This means that the position of the prey is estimated by the position of the three fittest agents in each iteration. As shown in the following equations, we can model the process of encircling the prey using the estimated position of the prey by:

\[
\tilde{D}_\alpha = |\tilde{C} \tilde{X}_\alpha(t) - \tilde{X}(t)|
\]

\[
\tilde{D}_\beta = |\tilde{C} \tilde{X}_\beta(t) - \tilde{X}(t)|
\]

\[
\tilde{D}_\gamma = |\tilde{C} \tilde{X}_\gamma(t) - \tilde{X}(t)|
\]

\[
\tilde{X}_1 = \tilde{X}_\alpha(t) - \tilde{A}_1 \tilde{D}_\alpha
\]

\[
\tilde{X}_2 = \tilde{X}_\beta(t) - \tilde{A}_2 \tilde{D}_\beta
\]

\[
\tilde{X}_3 = \tilde{X}_\gamma(t) - \tilde{A}_2 \tilde{D}_\gamma
\]

and finally updating the position of other subordinate wolves by:

\[
\tilde{X}(t + 1) = \text{round} \left( \frac{X_1 + X_2 + X_3}{3} \right)
\]

Note that since our design space is discrete, the \(\tilde{X}(t + 1)\) is rounded to the nearest integer.

Fig. 2 demonstrates the direction of the search based on the position of \(\alpha\), \(\beta\), and \(\gamma\) wolves. Algorithm. 2 shows the GWO pseudo code utilizing Eq. 2 to 5.

**C. BAYESIAN OPTIMIZATION**

Bayesian optimization [7] is a powerful method to find the extremum of an objective function that is expensive to evaluate. For instance, in hardware design space exploration problems, it normally takes hours to synthesize an OpenCL code using high-level synthesis tools. Bayesian Optimization assumes that the posterior probability of an objective function \(O\) for a given evidence sample data is proportional to the likelihood of samples multiplied by prior probability of \(O\).

Let us define \(x_i\) as the \(i^{th}\) sample and \(O(x_i)\) as the evaluation of the objective function for that sample. According to Bayes theorem, for a collection of \(t\) samples such as \(C_{1:t} = \{x_i, O(x_i)\}\):

\[
P(O|C_{1:t}) \propto P(C_{1:t}|O) \times P(O)
\]

where \(P(O|C_{1:t})\) denotes the posterior distribution. Since the posterior distribution accumulates beliefs about the objective function, it can be represented as a surrogate function.
which estimates the objective function \( O \). Note that in theory, Bayesian optimization assumes that the objective function is continuous. However, it is possible to estimate a surrogate function for a discrete objective function by fitting a continuous posterior function that passes through the discrete points of the objective function \( O \). This idea is also explored in [19].

In practice, the sampling process is considered noisy with a Gaussian Distribution \( N(0, \sigma) \) where \( \sigma \) denotes the noise that is incorporated with the observation. Likewise, the surrogate function is also considered to be a Gaussian Process. This means the surrogate function \( f_s \) returns mean and variance as opposed to a normal function which returns a scalar value.

\[
  f_s(x) = GP(m(x), K = k(x, x'))
\]  

Eq. 7 shows that the \( GP \) is constructed of a mean vector \( m \) and a covariance matrix \( K \) comprising of jointly Gaussian distribution between the sample population \( C_{1:T} \) and the next chosen sample \( x_{t+1} \). For this research, the squared exponential covariance kernel \( k \) is used to construct covariance matrix \( K \):

\[
  k(x_i, x_j) = \exp\left(-\frac{|x_i - x_j|^2}{2}\right)
\]  

Bayesian optimization is efficient in terms of exploration and exploitation since it incorporates prior belief using jointly Gaussian kernels to help with the acquisition direction. Moreover, the search is directed using the maximum probability of improvement (PI) function:

\[
  PI(x) = P \left( O(x^{\text{next}}) \geq O(x) \right)
  = \Phi \left( \frac{m(x) - O(x^{\text{next}})}{\sigma(x)} \right)
\]  

where \( O(\cdot) \) is the objective function sample, \( \Phi \) is the cumulative Gaussian distribution, and \( m(x) \) and \( \sigma(x) \) are the mean and variance of the samples respectively.

Algorithm 3 shows the pseudo-code of a BO method. For simplicity, it is divided into five steps. Step 1 is the guided search using the acquisition function. Step 2 and Step 3 show the noisy sampling and augmenting new samples to the sample population. Finally, Steps 4 and 5 show the Gaussian process update procedure.

After obtaining a surrogate model for objective function \( O \) (i.e surrogate function \( f_s(x) \)), we define the top-1 surrogate model suggestion as \( \text{argmax}(f_s(x)) \) and top-5 surrogate model suggestion as a set of \( x_i \) that produce the 5 largest local maximums of the surrogate function \( f_s(\cdot) \).

**D. HYBRID GWO-BO**

Assuming that the user who optimizes the objective function \( O \) wants to benefit from both GWO and BO algorithms, it is possible to combine those two algorithms. Thus, the user runs GWO on the dataset and passes the explored variables by the GWO search agents to BO to create a surrogate model. In this article, we show that GWO can provide qualified samples for Bayesian optimization and it can decrease the number of acquisitions of the objective function. Moreover, by changing the parameter \( \bar{a} \) in the GWO algorithm, it is possible to tune the exploration/exploitation balance for search agents.

The hybrid GWO-BO method can achieve superior performance compared to both GWO and BO methods alone, while constructing a model based on the samples explored by GWO which are considerably fewer than the number of samples used by the conventional BO model.

Algorithm 4 specifies the steps performed to search the design space using the hybrid GWO-BO method. First GWO collects the samples from the environment and saves them in the sample collections. Then the sample collection is passed to the BO to build the surrogate model using the Gaussian Process.

Section IV-D discusses in detail the results obtained with the hybrid GWO-BO method.

**E. JUSTIFICATION OF THE DESIGN SPACE EXPLORATION ALGORITHMS CHOICES**

We chose the GWO algorithm as it gives the user fine control over exploration/exploitation trade-offs using parameter \( \bar{a} \) as it will be discussed in Section IV-B. Moreover, GWO can be
easily combined with BO to create a surrogate model of the design space. BO is used merely for the purpose of surrogate modeling. The surrogate model can help us understand the interaction between the design space variables without performing expensive synthesis analysis. It will be shown in Section.V-C that a Bayesian surrogate model can represent the behavior of the design space more efficiently than regression models such as the LASSO (Least Absolute Shrinkage and Selection Operator) model. Finally, we proposed a hybrid GWO-BO model that improves the respective weaknesses of the individual GWO and BO algorithms. A hybrid GWO-BO model benefits from surrogate modeling of the BO, while it produces results that are at least as good as GWO.

F. BENCHMARK ALGORITHMS

In order to validate the results of our black-box optimization tool implementing the GWO-BO method, we used an OpenCL benchmark suite for FPGA targets called Spector [20]. The benchmark consists of ten benchmark functions that are commonly used in FPGA designs. The benchmark functions are OpenCL implementation of 1) Breadth-First Search Dense (BFS Dense) 2) Breadth-First Search Sparse (BFS Sparse) 3) Discrete Cosine Transform (DCT) 4) Finite-Impulse Response (FIR) filter 5) Histogram 6) Matrix Multiplication 7) Merge-Sort 8) 3D Normal Estimation 9) Sobel Filter and 10) Sparse Matrix-Vector Multiplication (SPMV).

Some design space variables are shared among the designs and some are specific to each design. For instance, work-items is a generic design variable that is related to the number of parallel threads in an OpenCL application. Likewise, work-group indicates the number of work-items to be used without shared memory, compute-units indicates the number of kernels launched on the device and SIMD shows the level of data-parallelism in the kernels. It is also possible to consider HLS directives such as unrolling as a binary variable that accepts 1 or 0 to enable or disable the functionality.

On the other hand, some variables are specific to each design. For example, the number of histograms in the histogram benchmark or the number of filter coefficients in the FIR benchmark.

Algorithm 4 Hybrid GWO-BO Algorithm

| Step 1. Perform GWO algorithm |
| Step 2. Save the wolves positions at each step $t$ in the sample collection: $C_{t:T} = \{x_{t:T-1}, O(x_{t:T-1})\}$ |
| Step 3. Save the corresponding objective function samples: $O(x_t)$ |
| for $t = 1, 2, 3 \ldots$ in (Sample collection) do |
| Step 4. Calculate new covariance matrix using eq. 8 |
| Step 5. Update $GP$ using eq. 7 |
| end |
| return $\text{argmax}(GP)$ |

IV. RESULTS

This section, characterizes, compares and discusses the design space exploration algorithms mentioned in Section. III.

A. HILL CLIMBING AND NON-CONVEXITY ANALYSIS

Introducing an algorithm that can perform optimally in non-convex design spaces is essential for researchers in this domain. Exploring a convex design space is fairly simple. For instance, the Hill climbing algorithm is very effective at finding the global optimum of a convex design spaces. However, in practice, it is not possible to guarantee the convexity of the design space or at least the landscape of the design space is unknown. Due to the unknown design landscape, exploration algorithms must be able to explore non-convex design spaces.

In this section, the Hill-Climbing algorithm is used solely as a metric to detect the non-convexity of the design space. This means that if the Hill-Climbing algorithm cannot find the global maximum, the design space is not convex. Note that observing failure of the Hill-Climbing algorithm to find the global optimum is a sufficient condition to confirm non-convexity. However, it is not a necessary condition. With this, if Hill-Climbing can find the global maximum, it is not guaranteed the design space is convex.

Table. 1 shows the execution of the Hill-Climbing algorithm over the benchmark functions mentioned in Section. III-F. According to the Hill-Climbing metric, all benchmark functions except the Breadth-First Search Sparse (BFS Sparse) are non-convex. For BFS Sparse, we cannot determine the convexity since the Hill climbing algorithm has successfully found its best variable choices.

| Benchmark Function | Hill Climbing Best Latency (ms) | Benchmark Best Latency (ms) [20] | Convexity |
|---------------------|--------------------------------|----------------------------------|-----------|
| BFS Dense           | 4.5051                         | 4.284                           | Non-convex |
| BFS Sparse          | 12.5151                        | 12.5151                         | N.D.¹     |
| DCT                 | 2.6458                         | 2.1916                          | Non-convex |
| FIR                 | 0.0566                         | $7.3 \times 10^{-4}$            | Non-convex |
| Histogram           | 1.7829                         | 1.5481                          | Non-convex |
| Mergesort           | 1.9555                         | 1.9269                          | Non-convex |
| Matrix Multiplication | 152.4176                     | 34.1980                         | Non-convex |
| Normal Estimation   | 6.6465                         | 6.2779                          | Non-convex |
| Sobel Filter        | 3.7522                         | 1.7190                          | Non-convex |
| Sparse Matrix Vector Multiplication | 0.1791   | 0.1661                          | Non-convex |

¹ Not determined.

As demonstrated in Table. 1, Hill climbing completely fails in determining the best latency of Matrix Multiplication and Sobel Filter benchmarks. Besides analyzing convexity, this experiment shows that designers should not ignore some simple incremental search algorithms such as Hill climbing.
in design space exploration problems. Although it is not possible to know the convexity prior to performing the experiments, it is always good to check whether Hill climbing is suitable. However, the essence of having a surrogate model to understand and analyze the interaction between the design variables remains unanswered with the Hill climbing method.

B. GREY WOLF OPTIMIZATION (GWO)

As shown in Section IV-A, Hill-Climbing can provide some initial understanding regarding the convexity of the design space and latency of the benchmark functions. However, it under-performs in more complicated benchmarks such as Matrix multiplication and Sobel Filter. The main reason is that Hill-Climbing is deterministic in its search scope, which means that it follows the direction of the largest increment in the objective function by evaluating the search agent’s neighbors. Furthermore, it is not possible to control the trade-offs between exploration and exploitation in Hill-climbing.

In contrast, GWO provides very flexible control over the exploration/exploitation trade-off. More specifically, the variable $\alpha$ in Eq. 2 controls this trade-off. For instance, when $\alpha$ is close to 2, the search agents are in their ultimate exploration mode. As $\alpha$ decreases from 2 to 1, the intensity of exploitation decreases. When $\alpha$ is 1, the search agents enter the exploitation mode and as $\alpha$ decreases from 1 to 0, the exploitation intensifies. The variable $\alpha$ is set to linearly decrease from 2 to 0 in the course of iterations to provide the best trade-off between exploration and exploitation.

Eq.5 uses a round to nearest integer to quantize the positions of the wolves since GWO is performed over a discrete search space. The rounding affects the exploitation in a way that if there is not a great expected improvement, the search agents tend to stay in their current position. This phenomenon will further decrease the number of iterations needed for design space exploration.

Fig. 3 demonstrates the effect of $\alpha$ on search agents’ exploration and exploitation trade-off for the Sobel Filter benchmark. The red triangles are the explored points by the agents while the blue circles show the design space. The balanced exploration and exploitation (i.e. $\alpha$ is linearly decreased from 2 to 0) can achieve better results in terms of finding the best throughput performance.

Table. 2 shows the performance of the GWO for the benchmark functions. The design space exploration tests were conducted for 100 trials and the average and standard deviation of latency has been reported here. Since GWO is a meta-heuristic algorithm that produces different results for different experiments, it is necessary to conduct at least 100 tests in order to understand the convergence behavior of the GWO.

GWO outperforms Hill climbing and finds near-optimum results for all the benchmarks. Specifically, for Sobel Filter and Matrix Multiplication benchmarks, GWO finds acceptable latency which was not possible to be found using Hill climbing method.

Although GWO provides interesting results for design space exploration, it is not enough to create a surrogate model to study the interaction between variables. Finding a surrogate model is the focus of the next sections.

C. BAYESIAN OPTIMIZATION (BO)

In BO, a surrogate model is constructed using statistical methods (as explained in III-C) on a set of random samples drawn from the environment. The quality of the surrogate model is proportional to the number of samples and the distribution of the samples.

Table.3 shows the result of design space exploration using the constructed Bayesian surrogate model. Once the surrogate model is built, it is possible to find its optimum point. Since it is relatively inexpensive to examine the surrogate model, it is possible to perform design exploration more efficiently by evaluating the top choices that the model suggests. For instance, in Table.3, the top 5 variable choices suggested by the model are also explored. The top-5 evaluation can achieve better performance in terms of finding more optimal design choices while it increases the number of samples drawn from...
TABLE 2. Design space exploration using Grey-Wolf Optimization for 100 trials.

| Benchmark Function       | Average Latency (ms) | Standard Deviation of Latency (ms) | Benchmark Latency (ms) [20] | Average Number of Explored Variable Sets |
|--------------------------|----------------------|-----------------------------------|-----------------------------|----------------------------------------|
| BFS Dense                | 4.335                | 0.056169                          | 4.28.4                      | 85                                     |
| BFS Sparse               | 12.5151              | 0                                  | 12.5151                     | 85                                     |
| DCT                      | 2.5827               | 0.15291                           | 2.1916                      | 35                                     |
| FIR                      | 1.9×10^{-3}          | 9.6×10^{-4}                       | 7.3×10^{-4}                 | 140                                    |
| Histogram                | 1.6634               | 0.093946                          | 1.5481                      | 120                                    |
| Mergesort                | 1.9456               | 0.013607                          | 1.9269                      | 230                                    |
| Matrix Multiplication    | **39.7234**          | 8.2397                            | 34.1980                     | 130                                    |
| Normal Estimation        | 6.493                | 0.14783                           | 6.2779                      | 110                                    |
| Sobel Filter             | **1.7819**           | 0.065844                          | 1.7190                      | 200                                    |
| Sparse Matrix Vector     | 0.16798              | 0.0028675                         | 0.1661                      | 100                                    |

TABLE 3. Design space exploration using Bayesian Optimization for 100 trials.

| Benchmark Function       | Average Latency of optimum point (ms) | Standard Deviation of optimum point (ms) | Average Latency of Top-5 model suggestions (ms) | Standard Deviation of Top-5 model suggestions (ms) | Benchmark Latency (ms) [20] | Number of Random Samples Evaluated |
|--------------------------|--------------------------------------|------------------------------------------|-----------------------------------------------|--------------------------------------------------|-----------------------------|-----------------------------------|
| BFS Dense                | 4.4632                               | 0.1401                                   | 4.3889                                       | 0.0619                                           | 4.28.4                      | 101                               |
| BFS Sparse               | 13.3025                              | 0.5139                                   | 12.7031                                      | 0.3525                                           | 12.5151                     | 101                               |
| DCT                      | 5.4859                               | 4.9634                                   | 2.9319                                       | 0.5295                                           | 2.1916                      | 42                                |
| FIR                      | 3×10^{-3}                            | 1×10^{-3}                                | 2.6×10^{-3}                                 | 4×10^{-4}                                        | 7.3×10^{-4}                 | 234                               |
| Histogram                | 6.7245                               | 8.5840                                   | 3.2551                                       | 4.8735                                           | 1.5481                      | 179                               |
| Mergesort                | 2.4989                               | 0.3899                                   | 2.0935                                       | 0.1882                                           | 1.9269                      | 306                               |
| Matrix Multiplication    | 41.7924                              | 13.0553                                  | **38.7072**                                 | **2.7853**                                       | 34.1980                     | 236                               |
| Normal Estimation        | 7.7216                               | 1.6686                                   | 6.6608                                       | 0.3319                                           | 6.2779                      | 139                               |
| Sobel Filter             | 1.9074                               | 0.2801                                   | 1.7846                                       | 0.0610                                           | 1.7190                      | 276                               |
| Sparse Matrix Vector     | 0.244                                | 0.0427                                   | 0.205                                        | 0.0380                                           | 0.1661                      | 148                               |

the system by 5 iterations which is insignificant compared to the number of samples needed to build the surrogate model. Although BO provides acceptable results, it almost always under-performs compared to GWO in both optimum mode and top-5 evaluation mode. Moreover, it usually needs more samples drawn from the environment. This can be explained by the quality of the random samples drawn from the design space. Since GWO performs a guided search in the design space environment, it can find the outliers and anomalies when searching for the optimum point. The hybrid model described in the next section helps to overcome the poor performance of BO by leveraging the strengths of GWO.

D. HYBRID GWO-BO

Table 4 shows the results of design space exploration using the hybrid GWO-BO method. The hybrid model outperforms both BO and GWO in finding better results in terms of average and standard deviation of latency. Note that in this experiment, the top-5 evaluation of the surrogate model is used to generate the results.
TABLE 4. Design space exploration using Hybrid GWO-BO for 100 trials.

| Benchmark Function       | Average Latency (ms) | Standard Deviation of Latency (ms) | Benchmark Latency (ms) [20] | Number of Random Samples Evaluated |
|--------------------------|----------------------|------------------------------------|-------------------------------|-----------------------------------|
| BFS Dense                | 4.329                | 0.0499                             | 4.284                         | 85                                |
| BFS Sparse               | 12.5151              | 0                                  | 12.5151                       | 85                                |
| DCT                      | 2.5891               | 0.1357                             | 2.1916                        | 35                                |
| FIR                      | $1.6 \times 10^{-4}$ | $8.6 \times 10^{-4}$              | $7.3 \times 10^{-4}$          | 140                               |
| Histogram                | 1.5652               | 0.04166                            | 1.5481                        | 120                               |
| Mergesort                | 1.9386               | 0.0138                             | 1.9269                        | 230                               |
| Matrix Multiplication    | 36.8913              | 2.5895                             | 34.1980                       | 130                               |
| Normal Estimation        | 6.4315               | 0.10566                            | 6.2779                        | 110                               |
| Sobel Filter             | 1.7434               | 0.0341                             | 1.7190                        | 200                               |
| Sparse Matrix Vector     | 0.1671               | 0.0022                             | 0.1661                        | 100                               |

Also note that since we are optimizing a hardware design problem, it is important to choose the algorithms that find the optimum point with fewer samples. For instance, each synthesis trial of an OpenCL design can take up to hours. Table 4 shows that GWO-BO method uses fewer number of samples to create the surrogate model.

V. DISCUSSION
A. MULTI-OBJECTIVE OPTIMIZATION AND PARETO FRONTIER

In the previous sections of this article, the considered design space methods exploration were tuned to maximize throughput. However, hardware design space exploration is normally a multi-objective problem where designers try to jointly optimize throughput and logic utilization. In multi-objective optimization, it is not possible to find a solution that simultaneously optimizes all objective functions, so the optimization involves finding the Pareto-frontier [21]. A Pareto-frontier is defined as a set of dominant solutions for the optimization problem. A dominant solution happens if none of the objective functions can improve in their scores unless it is by decreasing some other objective scores. A Pareto-frontier quantifies the interaction of different objectives. This section shows that having a surrogate model facilitates estimating the dominant solutions.

Here, we defined the objective space as the output of objective function $O$ mapped to normalized latency and normalized logic utilization. Also, note that the latency and logic utilization are the two objectives defined for this multi-objective optimization problem.

Revisiting Fig. 1, the design space explorer makes a query to the synthesizer and retrieves the synthesis information. This information includes the area utilization of the system. Fitting Gaussian processes are not complicated. Thus, it is possible to create a second surrogate model for area utilization as well. Furthermore, having surrogate models for throughput and area consumption, estimating the Pareto-frontier on the surrogate models is inexpensive.

Fig. 4 shows the comparison between the actual Pareto-frontier and estimated Pareto-frontier for the two most challenging design spaces (Matrix Multiplication and Sobel Filter). The estimated Pareto-frontier is generated using the surrogate models for throughput and area utilization. Note that the surrogate models are made using the hybrid GWO-BO method. Although estimated Pareto-frontiers are very close to the actual Pareto-frontier, they are not identical. This is because the surrogate models are estimating the behavior of the system statistically.

B. QUALITATIVE COMPARISON OF DESIGN SPACE EXPLORATION METHODS

To study the quality of the design space exploration methods introduced previously, a fitness factor can be defined as follows:

$$F^{DSE} = 1 - \left( \frac{|l^{DSE}_{avg} - l^{benchmark}|}{l^{min}} \right)$$

where $F^{DSE}$ is the fitness of the design space exploration method, $l^{DSE}_{avg}$ is the average latency that is achieved with the method, and $l^{benchmark}$ is the latency of the benchmark. When the fitness parameter $F^{DSE}$ is closer to 1, the quality of the design space exploration is better. In other words, the fitness factor $F^{DSE}$ is the normalized latency that is found by a given design space exploration method.
Also, it is worth mentioning that the optimum solution (i.e., $p_{\text{benchmark}}^{\text{min}}$) is known to us when computing the fitness factor according to [20].

Fig. 5 compares the fitness factor of the three design space exploration methods explained in the previous sections for each benchmark function. It shows that GWO (grey bars) can find near-optimum results. Despite the good search performance of GWO, it cannot provide a surrogate model of the design space objective function. Furthermore, the quality of the results obtained with Bayesian-optimization (green bars) are not as good as those obtained with GWO in several cases (e.g. FIR, Histograms, SPMV, and Mergesort). Finally, a Hybrid GWO-BO model performs at least as well as GWO and also benefits from the surrogate modeling of the design space. Hence, these results show that the combination of GWO and BO always chooses the near-optimum settings for the benchmark functions.

**FIGURE 4.** Red triangles in (a) and (c) are the Pareto-frontier estimated by surrogate models for throughput and area utilization. Red triangles in (b) and (d) are the actual Pareto-frontier of the design space according to benchmark [20].

**FIGURE 5.** Qualitative comparison of design space exploration methods. The vertical axis shows the fitness $F^{DSE}$. Higher fitness factors correspond to better solutions.

### C. COMPARISON WITH PREVIOUS WORKS

In this section, we compare our results with previous works. In [22], it is shown that a Random Search can be more efficient for parameter tuning than a grid search. Thus, to demonstrate the effectiveness of the methods proposed in this paper, we first compare our results with a random search strategy to show that we can find better parameter settings for an OpenCL high-level synthesis workflow. Thus, a random search method was implemented and applied to the dataset used in this paper.

Moreover, we also compared our results with the LASSO (Least Absolute Shrinkage and Selection Operator) regression model introduced in [20]. In both cases, GWO-BO finds more qualified optimum solutions faster.

Table. 5 summarizes this comparison. The random search is done over 100 trials and the average latency as well as standard deviation of the best solution obtained in the trials.
TABLE 5. Comparison with previous works.

| Benchmark Function         | Random Search [22] Average \(^1\) Latency (ms) | Random Search [22] Standard Deviation of Latency \(^1\) (ms) | LASSO model (ms) [20] | This work\(^2\) Average Latency (ms) | This work\(^2\) Standard Deviation of Latency (ms) |
|---------------------------|-----------------------------------------------|-------------------------------------------------------------|-----------------------|------------------------------------|-----------------------------------------------|
| BFS Dense                 | 4.372                                         | 0.05304                                                    | 4.3825                | 4.329                             | 0.0499                                       |
| BFS Sparse                | 13.371                                        | 0.5449                                                     | 13.2857               | 12.5151                           | 0                                             |
| DCT                       | 2.904                                         | 1.485                                                      | 2.9522                | 2.5891                            | 0.1357                                       |
| FIR                       | 1.8 × 10^{-3}                                 | 1.03 × 10^{-3}                                             | 2.4 × 10^{-3}         | 1.6 × 10^{-4}                     | 8.6 × 10^{-4}                                |
| Histogram                 | 1.6632                                        | 0.1232                                                     | 1.5715                | 1.5652                            | 0.0416                                       |
| Mergesort                 | 1.9955                                        | 0.0685                                                     | 2.0441                | 1.9386                            | 0.0138                                       |
| Matrix Multiplication     | 40.9993                                       | 8.8535                                                     | 40.6412               | 36.8913                           | 2.5895                                       |
| Normal Estimation         | 6.5449                                        | 0.2311                                                     | 6.4763                | 6.4315                            | 0.10566                                      |
| Sobel Filter              | 1.8448                                        | 0.1145                                                     | 1.9331                | 1.7434                            | 0.0341                                       |
| Sparse Matrix Vector      | 0.1729                                        | 0.0056                                                     | 0.1798                | 0.1671                            | 0.0022                                       |

\(^1\) 100 trials, number of random samples is equal to Table. 4
\(^2\) Using hybrid GWO-BO method

are reported in the table. Furthermore, in order to make a fair comparison, the number of random samples in each trial is equal to the numbers reported in Table. 4 for the GWO-BO algorithm. Note that GWO-BO algorithm produces better results, both in terms of average latency and its standard deviation in all benchmark functions.

Table. 5 also includes the results obtained with the LASSO model over the whole design space. We used the exact LASSO model of [20]. Although, all the design space information is incorporated in the LASSO model, it did not perform better than the GWO-BO model in any of the benchmark functions. We suspect that regression models such as LASSO are not suitable to model the optimum behavior, since they intrinsically smooth the outlier information.

Also, it is noteworthy that in Table. 5, there are some extreme cases where GWO-BO offers significant advantages over the LASSO model and the random search. For instance, in BFS sparse, GWO-BO can always find the global maximum with a standard deviation of zero, while the random search and LASSO model fail to find it. Another significant example is the FIR benchmark, where the standard deviation obtained with the GWO-BO search is significantly smaller than the results obtained with random search. Likewise, in the Matrix Multiplication benchmark, the GWO-BO method can find significantly better latency than the random search or LASSO model.

D. OTHER RELATED WORKS

GWO has recently been used for other exploration problems. For instance, in [23], the authors used the GWO algorithm for evolving Convolutional Neural Network Long Short-Term Memory (CNN-LSTM) for time series analysis. They showed that GWO can produce significantly better results than other meta-heuristic methods. Moreover, as discussed in Section IV-B, they manipulated the \(\vec{a}\) parameter of the GWO algorithm to reach the best trade-off between exploration and exploitation.

Similarly, in [24], the authors used the GWO algorithm to explore and find the best hyper-parameter settings for LSTM-based language models. Their experimental results show that the GWO algorithm with 15 search agents can find the global optimum of the search space. Also, in [25], the authors used a modified GWO algorithm to solve the problem of workflow scheduling in the cloud. Their results show that the modified GWO algorithm can outperform the common scheduling approaches in terms of power consumption and cost.

VI. CONCLUSION

This paper describes the implementation of a design space exploration tool to optimize OpenCL kernels for FPGA targets. Three design space exploration methods (GWO, BO, GWO-BO) have been implemented. Grey Wolf Optimization (GWO) is a meta-heuristic that produces reasonable performance for design space search problems, but it does not provide a surrogate model of the system. Bayesian Optimization (BO) is another well-known method that can provide a surrogate model of the system, however, it does not perform as well as GWO. This paper ultimately proposes a novel hybrid GWO-BO method that combines the BO and GWO in a way that the BO uses the samples explored by GWO to create a surrogate model. According to the
benchmark tests, the hybrid method outperforms GWO and BO. Finally, we suggest that the surrogate model produced by the hybrid GWO-BO model can also be used to estimate the Pareto-frontier for multi-objective area/throughput optimization.

ACKNOWLEDGMENT

The authors would like to thank Dr. Melanie Segado and Yuvar Korkian for their valuable comments.

REFERENCES

[1] E.-G. Talbi and T. Muntean, “Hill-climbing, simulated annealing and genetic algorithms: A comparative study and application to the mapping problem,” in Proc. 26th Hawaii Int. Conf. Syst. Sci., vol. 2, Jan. 1993, pp. 565–573.

[2] C. Audet and W. Hare, “Derivative-Free and Blackbox Optimization,” Cham, Switzerland: Springer, 2017, doi: 10.1007/978-3-319-68913-5.

[3] B. Reagen, I. M. Hernandez-Lobato, R. Adolf, M. Gelbart, P. Whatmough, G.-Y. Wei, and D. Brooks, “A case for efficient accelerator design space exploration via Bayesian optimization,” in Proc. IEEE/ACM Int. Symp. Low Power Electron. Design (ISLPED), Jul. 2017, pp. 1–6.

[4] P. Zhao and B. Yu, “On model selection consistency of lasso,” J. Mach. Learn. Res., vol. 7, pp. 2541–2563, Nov. 2006.

[5] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan, “A fast and elitist multiobjective genetic algorithm: NSGA-II,” IEEE Trans. Evol. Comput., vol. 6, no. 2, pp. 182–197, Apr. 2002.

[6] B. C. Schafer, T. Takenaka, and K. Wakabayashi, “Adaptive simulated annealer for high level synthesis design space exploration,” in Proc. Int. Symp. VLSI Design, Autom. Test, Apr. 2009, pp. 106–109.

[7] E. Brochu, V. M. Cora, and N. de Freitas, “A tutorial on Bayesian optimization of expensive cost functions, with application to active user modeling and hierarchical reinforcement learning,” 2010, arXiv:1012.2599.

[8] A. Mehrabi, A. Manocha, B. C. Lee, and J. D. Sorin, “Bayesian optimization for efficient accelerator synthesis,” ACM Trans. Archit. Code Optim., vol. 18, no. 1, pp. 1–25, Jan. 2021.

[9] C. Lo and P. Chow, “Model-based optimization of high level synthesis directives,” in Proc. 26th Int. Conf. Field Program. Log. Appl. (FPL), Aug. 2016, pp. 1–10.

[10] S. Mirjalili, S. M. Mirjalili, and A. Lewis, “Grey wolf optimizer,” Adv. Eng. Softw., vol. 69, pp. 46–61, Mar. 2014.

[11] L. P. Kaelbling, M. L. Littman, and A. W. Moore, “Reinforcement learning: A survey,” J. Artif. Intell. Res., vol. 4, no. 1, pp. 237–285, Jan. 1996.

[12] M. de Prado, A. Mundy, R. Saeed, M. Denna, N. Pazos, and L. Benini, “Automated design space exploration for optimised deployment of DNN on arm cortex—A CPUs,” IEEE Trans. Comput.-Aided Design, vol. 9, no. 12, p. 2200, Dec. 2020.

[13] A. Ghaffari and Y. Savaria, “CNN2Gate: An implementation of convolutional neural networks inference on FPGAs with automated design space exploration,” Electronics, vol. 9, no. 12, p. 40, Dec. 2020.

[14] V. Minh, A. P. Badia, M. Mirza, A. Graves, T. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu, “Asynchronous methods for deep reinforcement learning,” in Proc. Int. Conf. Mach. Learn., 2016, pp. 1928–1937.

[15] H.-Y. Liu and D. P. Carloni, “On learning-based methods for design-space exploration with high-level synthesis,” in Proc. 50th Annu. Design Autom. Conf. (DAC), 2013, pp. 1–7.

[16] G. Beltrame, L. Fossati, and D. Sciuto, “Decision-theoretic design space exploration of multiprocessor platforms,” IEEE Trans. Comput.-Aided Design Integr. Circuits Syst., vol. 29, no. 7, pp. 1083–1095, Jul. 2010.

[17] E. B. Pinto and C. Doerr, “A simple proof for the usefulness of crossover in black-box optimization,” in Proc. Int. Conf. Parallel Problem Solving Nature. Cham, Switzerland: Springer, 2018, pp. 29–41, doi: 10.1007/978-3-319-99259-4_3.

[18] D. H. Wolper and W. G. Maceready, “Free lunch theorems for optimization,” IEEE Trans. Evol. Comput., vol. 1, no. 1, pp. 67–82, Apr. 1997.

[19] P. Luong, S. Gupta, D. Nguyen, S. Rana, and S. Venkatesh, “Bayesian optimization with discrete variables,” in Proc. Australas. Joint Conf. Artif. Intell. Cham, Switzerland: Springer, 2019, pp. 473–484, doi: 10.1007/978-3-030-35288-2_38.

[20] Q. Gautier, A. Aithoff, P. Meng, and R. Kastner, “Spector: An OpenCL FPGA benchmark suite,” in Proc. Int. Conf. Field-Program. Technol. (FPT), Dec. 2016, pp. 141–148.

[21] P. Ngatchou, A. Zarei, and A. El-Sharkawi, “Pareto multi objective optimization,” in Proc. 13th Int. Conf., Intell. Syst. Appl. Power Syst., Nov. 2005, pp. 84–91.

[22] J. Bergstra and Y. Bengio, “Random search for hyper-parameter optimization,” J. Mach. Learn. Res., vol. 13, no. 2, pp. 1–25, 2012.

[23] H. Xie, L. Zhang, and C. P. Lim, “Evolving CNN-LSTM models for time series prediction using enhanced grey wolf optimizer,” IEEE Access, vol. 8, pp. 161519–161541, 2020.

[24] B. Z. Aufa, S. Suyanto, and A. Arifianito, “Hyperparameter setting of LSTM-based language model using grey wolf optimizer,” in Proc. Int. Conf. Data Sci. Appl. (ICoDSA), Aug. 2020, pp. 1–5.

[25] A. Mohammadzaehdeh, M. Masardi, F. S. Gharechopogah, and A. Jafarian, “Improved chaotic binary grey wolf optimization algorithm for workflow scheduling in green cloud computing,” in Evolutionary Intelligence. Springer, Sep. 2020, pp. 1–29, doi: 10.1007/978-0-20-00479-5.

ALIREZA GHAFFARI received the B.Sc. degree from Amirkabir University of Technology (Tehran Polytechnic), Iran, and the M.Sc. degree from Laval University, Canada, both in electrical engineering. He is currently pursuing the Ph.D. degree with Polytechnique Montréal. He has a wide range of industrial and research experience in firmware design, FPGA, hardware acceleration, and machine learning. Recently, he is more focused on hardware/software acceleration of compute-intensive algorithms.

YVON SAVARIA (Fellow, IEEE) received the B.Eng. and M.Sc.A. degrees in electrical engineering from École Polytechnique Montréal, in 1980 and 1982, respectively, and the Ph.D. degree in electrical engineering from McGill University, in 1985. Since 1985, he has been with Polytechnique Montréal, where he is currently a Professor with the Department of Electrical Engineering. He is also affiliated with Hangzhou Innovation Institute, Beihang University. He has been working as a Consultant or was sponsored for carrying out research with Bombardier, Buspass, CNRC, Design Workshop, Dolphin, DREO, Ericsson, Genesis, Genum, Huawei, Hyperchip, Intel, ISR, Kaloom, LTRIM, Miranda, MiroTech, Nortel, Octasic, PMC-Sierra, Space Codesign, Technoport, Thales, Tundra, and Wavelite. He has carried out work in several areas related to microelectronic circuits and microsystems, such as testing, verification, clocking methods, defect and fault tolerance, effects of radiation on electronics, high-speed interconnects and circuit design techniques, CAD methods, reconﬁgurable computing and applications of microelectronics to telecommunications, aerospace, image processing, video processing, radar signal processing, and the acceleration of digital signal processing. He is involved in several projects related to embedded systems in aircraft, radiation effects on electronics, asynchronous circuit design and testing, green IT, wireless sensor networks, virtual networks, software-deﬁned networks, machine learning, embedded ML, computational efﬁciency, and application-speciﬁc architecture design. He holds 18 patents, has published 180 journal articles, and 480 conference papers. He was the Thesis Advisor of 170 graduate students who completed their studies. He is a member of the Executive Committee of the Regroupement Stratégique en Microélectronique du Québec (RESMIQ), the Ordre des Ingénieurs du Québec (OIQ), and the CMC Microsystems Technical Advisory Committee. In 2001, he was awarded the Tier 1 Canada Research Chair (www.chairs.gc.ca) on the designs and architectures of advanced microelectronic systems that he held until June 2015. He also received the Synergy Award of the Natural Sciences and Engineering Research Council of Canada, in 2006. He was the Program Co-Chair of NEWCAS’2018 and the General Chair of NEWCAS’2020. Since June 2019, he has been the NSERC-Kaloom-Intel-Noviflow (KIN) Chair Professor.