Automating Optimization of Reconfigurable Designs

Maciej Kurek, Tobias Becker, Thomas C.P. Chau and Wayne Luk
Department of Computing, Imperial College London

***Abstract*—We present Automatic Reconfigurable Design Efficient Global Optimization (ARDEGO), a new algorithm based on the existing Efficient Global Optimization (EGO) methodology for automating optimization of reconfigurable designs targeting Field-Programmable Gate Array (FPGA) technology. It is a potentially disruptive design approach: instead of manually improving designs repeatedly but without understanding the design space as a whole, ARDEGO users follow a novel approach that: (a) automates the manual optimization process, significantly reducing optimization time and (b) does not require the user to calibrate or understand the inner workings of the algorithm. We evaluate ARDEGO using two case studies: financial option pricing and seismic imaging.**

Field-Programmable Gate Arrays (FPGAs), and other reconfigurable computing devices, can provide high computational performance but their productive adoption has been hampered due to the long hardware design cycle. Furthermore optimizing design parameters requires tremendous effort in analyzing the application to create models and benchmarks. Examples of such parameters are clock frequency or numerical representation [1], [2], [3]. Ideally, those design parameters would be *automatically* optimized using an *calibration free* algorithm without manual setup or tuning, like exhaustive search or hill-climbing. However, this is impractical as these schemes require hundreds of data points, while even a single test design can take hours to build. Some optimization techniques like mathematical programming or gradient descent make assumptions about the optimized system, like convexity or continuity of the underlying problem, which have to be verified by the designer, thus the *calibration free* principle does not hold. To address these issues the Efficient Global Optimization (EGO) algorithm [4] is used as the basis for our new methodology. This approach does not require the designer to tune the algorithm or to analytically study and model the underlying effects. Our contributions are:

- A novel *calibration free* Automatic Reconfigurable Design Efficient Global Optimization (ARDEGO) algorithm that offers automatic optimization of reconfigurable designs. (Section I.)
- An evaluation of ARDEGO using two case studies: a quadrature design for financial computation [1], and a Reverse Time Migration (RTM) design for seismic imaging with multiple parameters [3]. (Section II.)

The process of automatic optimization is illustrated in Figure 1. The designer starts by describing the design and writing benchmarks. Benchmarks evaluate the reconfigurable design's parameters, which is a time-consuming process and often involves hardware generation and benchmark execution. The output of a benchmark is a scalar performance measure y called *fitness*: execution time, energy or any other target quality. In the case of reconfigurable designs, the *fitness* function f represents the benchmark, and the vector $\mathbf{x}$ is the parameter setting within the design space $\mathbf{x} \in \mathcal{X}$ and D dimensions (parameters) with $f(\mathbf{x}) = y$. Once the benchmark's code and the design description are ready, the designer specifies the design space and the design constraints. The space specifies the architecture and the physical settings of the FPGA design. A design may build successfully or violate one of several constraints, such as area or timing, terminating with an appropriate exit code t [5]. This exit code is used to determine whether a design is *valid* or not.

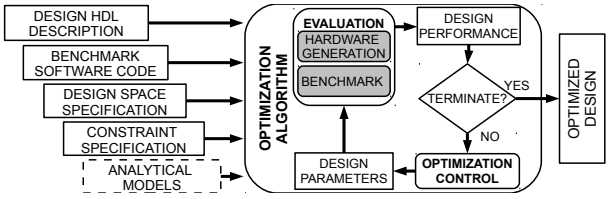

Figure 1: Optimization approach.

Using feedback from design evaluation a *surrogate model* can be constructed and used by an automatic optimization algorithm. A *surrogate model* is an approximation technique that is used when the behaviour of the underlying problem is not known or too expensive to measure, like in the case of reconfigurable hardware design. Speeding up time-consuming high-level synthesis using surrogate modeling based on *fitness* inheritance has been explored in [6]. *Surrogate models* are also used in Machine Learning Optimizer (MLO), developed to optimize parameters of reconfigurable designs [5].

Gaussian Process (GP) is a supervised learning method capable of *regression* [7], often used as basis of the *surrogate model*. *Regression* is an approach to model dependency between variable changes and scalar output of a system. GPs are often used when a predictive distribution is required instead of a scalar estimate. The goal is to obtain the distribution of the function $\hat{f}$ at input $\mathbf{x}$ given a set of input vectors $\mathbb{X}$, the associated past observations $f(\mathbb{X})=\mathbb{Y}$, and the *kernel function* $k(\mathbf{x}, \mathbf{x}')$. *Kernel functions* are used to transform the original space to a different space, possibly yielding better predictive power. The standard deviation estimate $\sigma(\mathbf{x})$ is often interpreted as measure of the uncertainty of the prediction at point $\mathbf{x}$. For constrained optimization problems

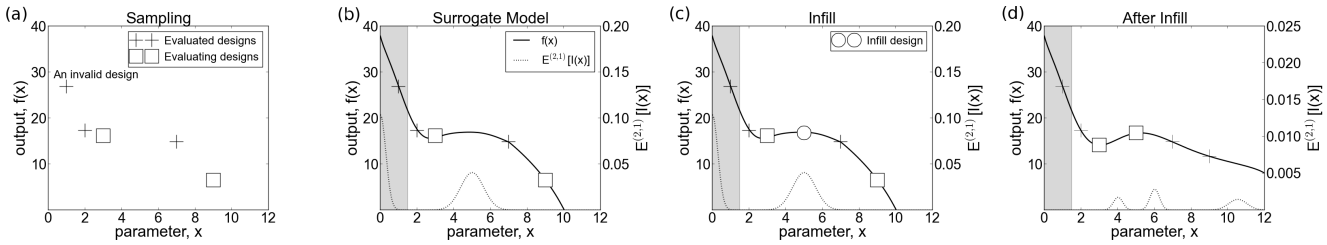

Figure 2: ARDEGO iteration with three worker nodes (P = 3), the algorithm starts with sampling (a) after which it moves into the infill state. At first the *surrogate model* is constructed, with two designs being evaluated (b). With one worker node free, the current *surrogate model* is used to search for infill design (c). In (d) a worker has finished evaluation and a new infill has to be found, the process repeats.

surrogate models can include classifiers, particularly Support Vector Machines (SVMs) [5], [8]. In such a case, besides modeling the *fitness function* the *surrogate model* determines which, if any, constraints are likely to fail. The goal of *classification* is to construct a decision function d allowing prediction of a class label $d(\mathbf{x}) = t$ for an unseen input $\mathbf{x}$. The decision function is constructed based on the observed data $\mathbb{X}, \mathbb{T}$ and the kernel function $k(\mathbf{x}, \mathbf{x}')$. The target labels associated with observations at $\mathbb{X}$ are denoted as $\mathbb{T}$. Extension of the *surrogate model* with a *classifier* allows for pruning of the design space. This is crucial when the design space is large and the designs take a long time to evaluate.

The Expected Improvement (EI) metric [4] tells how much a design $\mathbf{x}$ is likely to improve over the best currently found designs fitness $f_{best} = \min(\mathbb{Y})$. Given the mean estimate $\hat{f}(\mathbf{x})$ and standard deviation $\sigma(\mathbf{x})$ by the *surrogate model* GP *regression*, improvement of $\mathbf{x}$ over f_{best} is defined as $I(\mathbf{x})$ and its expected value $E[I(\mathbf{x})]$ is used as the EI metric. This metric is used to repeatedly evaluate designs which are expected to yield highest improvement over the current best solution. The concept is used in the EGO algorithms [4], [8]. The EGO algorithm was is for parallel systems with P worker nodes [4]. When developing automated optimization techniques for reconfigurable designs, we face two challenges: 1) the algorithm should not require manual calibration and 2) the algorithm should be capable of constructing an initial *surrogate model* even if the number of designs not violating constraints is small relative to the design space.

I. ARDEGO Approach

The algorithm is designed to offer automatic *calibration free* design optimization as outlined in Figure 1. It is based on a *surrogate model* with an integrated SVM classifier. The key steps of the ARDEGO algorithm are illustrated in Figure 2. The algorithm starts to build the initial *surrogate model* with sampling of the design space, generating hardware for these design samples and evaluating their fitness. After the initial *surrogate model* is constructed, an iterative process follows where the model is refined with infill. The goal of infill is to find the designs that are most likely to improve over the currently best found design. Hardware is then built for them, their *fitness* is evaluated and the *surrogate model* updated accordingly. The two challenges mentioned earlier are addressed by ARDEGO in the following ways:

1) ARDEGO is based on EGO and has no parameters of its own that require calibration.
2) A novel *adaptive sampling plan* addresses the issue of design spaces with a small number of *valid* designs.

The ARDEGO algorithm follows the pseudocode illustrated in Figure 3. The algorithm is based on parallel EGO [4], constrained EGO [8] and the new *adaptive sampling plan*. The optimization procedure itself is a sequential process that is carried out by a control node. The control node invokes P worker nodes which can build and evaluate designs in parallel. Whenever a worker node finishes evaluation of a design, the control node searches for new infill designs. At any given time μ nodes are busy and λ are idle. The goal during infill is to find the set of λ most promising designs $\mathbf{X}_{idle} = [\mathbf{x}_1, \mathbf{x}_2, .., \mathbf{x}_\lambda]$, with highest $E[I^{(\mu,\lambda)}(\mathbf{X}_{idle})]$. The infill search on the control node does not block the optimization on the worker nodes and multiple worker nodes can finish evaluation at similar time. The termination condition is checked during infill whenever a worker node finishes evaluation: it is checked whether the time budget allocated for optimization has been exceeded.

```
1 Evaluate the always valid design;                    // Adaptive
2 Sample designs using Latin hypercube;                // Sampling
3 Sample random designs, do not evaluate designs which are
  predicted to fail any constraints. After every sample update
  the SVM model;
4 while not Termination condition do                   // Infill
5     [non blocking] Collect results of evaluated designs.
      Update the surrogate model model. Update λ and μ;
6     Generate the next λ infill points by max
      E[I^(μ,λ)(X_idle)] using a global optimizer (e.g. CMA-ES
      [9]). Send them to worker nodes for evaluation;
```

Figure 3: ARDEGO, based on parallel EGO [4].

The *surrogate model* consists of a GP regressor and a SVM *classifier*. The GP assesses the standard deviation estimate $\sigma(\mathbf{x})$ and the fitness $\hat{f}(\mathbf{x})$ of parameter configurations $\mathbf{x}$ that have not been evaluated. The SVMs predicts the exit code t for such configurations (i.e. whether the design is valid or not). *Regression* and *classification* are correspondingly based on the results of previous hardware generations and

benchmark executions aggregated in $\mathbb{X}, \mathbb{Y}$ and $\mathbb{T}$. The GP regressor uses the anisotropic Gaussian *kernel function*, which allows learning of the impact of different parameters on the design space [7]. The Gaussian *kernel function* is used for the SVM as it has universal approximation properties [8].

A. Adaptive Sampling Plan

The algorithm involves a two-stage *adaptive sampling plan* to allow good coverage of the *valid* space. Initially, if available, an *always valid* design is evaluated. It is a design representing the most basic configuration which the designer is certain will generate successfully. This allows ARDEGO to localize the region of space which does not violate any constraints. This is especially important if the region is relatively small. The sampling plan has two stages, a Latin hypercube sampling of $5 \times D$ designs followed by a random sampling of $5 \times D$ designs. This gives a total of 10 designs per dimension as recommended by [4], [8]. A Latin hypercube plan offers good space filling qualities which improve performance of the optimization algorithm [10] while the subsequent random sampling is mainly used to sample more *valid* designs for the GP *regression*. This sampling methodology allows for *regression* in cases when the *valid* region is small relative to the design space.

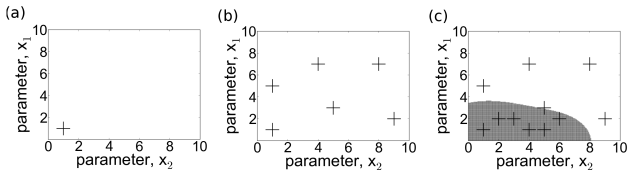

Figure 4: *Adaptive sampling plan*, gray area indicates *valid* region.

A visualization of the *adaptive sampling plan* is presented in Figure 4. Initially the *always valid* design is evaluated (a), followed by a 5-design Latin hypercube sampling (b). The subsequent random sampling plan (c) evaluates 5 randomly chosen designs from the *valid* area, tuning its shape. Figure 5 contains result of sampling and initial *classification* in the three other sampling plans. Although the identified *valid* area has similar shape, the number of designs available for *regression* is severely limited.

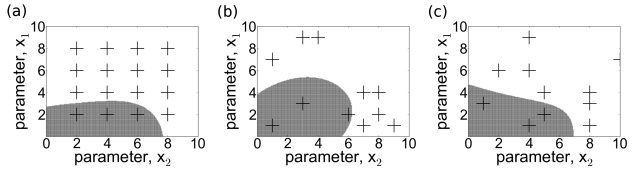

Figure 5: Grid, random and Latin Hypercube sampling plans.

B. Infill

This step consists of search over the design space for λ designs that are most likely to improve f_{best} and of their subsequent evaluation on the idle worker nodes. In the meanwhile μ designs $\mathbf{X}_{busy} = [\mathbf{x}_\lambda, \mathbf{x}_{\lambda+1}, .., \mathbf{x}_{\lambda+\mu}]$ are being evaluated on the busy nodes. The $E[I^{(\mu,\lambda)}]$ is used as the EI metric, its estimation with *sims* simulations is presented in Figure 6. It consists of estimation of the *fitness* and uncertainty, which is subsequently used to calculate $E[I^{(\mu,\lambda)}]$ using Monte Carlo techniques. The $Y_{(\omega)}(\mathbf{x})$ is a Gaussian random number conditioned on the past observations $\mathbb{X}$,$\mathbb{Y}$ i.e. $Y_{(\omega)}(\mathbf{x}) \sim \mathcal{N}(\hat{f}(\mathbf{x}), \sigma(\mathbf{x}))$, the point distribution returned by the GP *regression*. The *classification* is incorporated by returning a Gaussian distribution with f_{best} mean and standard deviation 0 for the designs which are predicted to violate some constraints. Referring to Figure 6, for those designs $E[I^{(\mu,\lambda)}](\mathbf{X}) = 0$.

```
1 f̂(X), σ(X) = surr_model_pred(X);
2 sum=0;
3 for i ∈ [0, 1, .., sims] do
4     Y^μ_(ω), Y^λ_(ω) ~ N(f̂(X), σ(X));
5     sum += [min(f_best, Y^μ_(ω)) − min(Y^λ_(ω))]^+;
6 sum = sum ÷ sims;
```

Figure 6: $E_{MC}I^{(\mu,\lambda)}(\mathbf{X}_{idle})$. For notation purposes busy and idle designs are aggregated in $\mathbf{X} = [\mathbf{x}_1, \mathbf{x}_2, .., \mathbf{x}_{\lambda+\mu}]$.

II. Evaluation

Two case application studies are used to evaluate ARDEGO optimization time. A quadrature-based financial design with customizable precision [1] and a high performance RTM design with seven parameters [3]. Both involve complex design choices, and have non trivial constraints. ARDEGO is compared with MLO [5] and hill climbing. Hill climbing is a trivial optimization algorithm which can deal with mixed continuous and discrete design spaces, and makes no assumptions about the *fitness function*. To make the optimization experiments repeatable, data on the application case studies are collected prior to the experiments. The total optimization time is measured, the results are averaged over 5 experiments. The optimized application designs target a Maxeler Max 3 system with a Xilinx Virtex-6 XC6VSX475T FPGA. The control and worker nodes consist of high performance Intel Xeon x5650 (32 nm, 6 cores, 2.67GHz) CPUs.

1) Quadrature-based Financial Design: In [1] the authors present a precision optimization methodology for a quadrature-based numerical integration solver for financial option pricing on reconfigurable devices. The design has two benchmarks measuring the throughput and energy consumption of a given configuration. The goal is to find the design offering the highest throughput or the lowest energy consumption given a required minimum accuracy ϵ_{rms} by optimizing three parameters. The three parameters are mantissa width m_w of the floating point operators, the number of computational cores $cores$, and the density factor d_f which specifies the density of quadratures used for integral estimation. ARDEGO is evaluated for three different ϵ_{rms}, which influence the number of *valid* designs. The design space $\mathcal{X}$ spans 18,000 configurations, with an

Table I: Total design optimization time given in hours. H.C. stands for hill climbing. MLO follows [5]. Q. En. is abbreviation for Quadrature-based Financial Design energy benchmark, Q. Th. is the corresponding throughput benchmark.

	Optimization Algorithm					
Design	MLO [5]	ARDEGO P				H.C.
		1	2	4	6	
Q. En. (ϵ_{rms} 0.01)	113	81	53	30	19	226
Q. En. (ϵ_{rms} 0.1)	139	120	54	35	27	273
Q. Th. (ϵ_{rms} 0.01)	122	109	43	30	26	328
Q. Th. (ϵ_{rms} 0.1)	174	131	72	41	39	502
RTM	*	285	197	123	97	5350

average hardware generation time of around 2 hours. The total optimization time using the application specific optimization methodology, including hardware generation and benchmark execution, is 198 hours. Depending on e_{rms}, the optimal design can be between 10-100 times faster and up to 200 times more power efficient than the most basic design.

As seen in Table I, in all of the cases P offers significant speedup in optimization time, yet even in when $\mathrm{P} = 1$ ARDEGO is substantially faster than MLO. The hill climbing algorithm offers very poor performance, in the worst case it takes twice the amount of time which is needed by the manual approach (502 vs. 198 hours). The advantage of the manual approach is certainty in finding the optimal design, although at a high effort [1]. Although ARDEGO does not guarantee that the optimal configuration is found, it can substantially reduce the optimization time. As seen in Table I, the optimization can be sped up by multiple nodes P, but the scalability requires further investigation. The ARDEGO overhead time, optimization time excluding hardware generation, is around an hour for any ϵ_{rms}, including infill search and *surrogate model* training.

2) Reverse Time Migration Design: In [3] the designer faces a problem of optimizing seven parameters of a high performance RTM design and there are nearly 27 million possible parameter combinations. The RTM design is used for seismic imaging to detect terrain images of geological structures. The design involves stencil computation, and most of the parameters are related to balancing communication and computation ratios as well as controlling the internal architectural settings such as parallelism and numerical precision to find an optimal design. The parameters explored are blocking ratios in x and y dimensions (α and β), bit-width optimization ratio B, arithmetic operation transformation ratio T, and kernel and dimension parallelism, P_{dp}, P_{knl} and P_t. Hardware generation time takes up to 9 hours for a single design. An analytical model has been developed to enable optimization of the design involving memory architectures, precision optimization, computation transformation, and design scalability [3]. The optimized design is over 100 times faster than the basic configuration.

As observed in Table I, the hill climbing optimization algorithm requires unrealistic amount of time to find the optimal design. This is because of the dimensionality of the problem, where hundreds of designs have to be evaluated. MLO is not capable of optimization of high dimensional designs, even when upgraded with the *adaptive sampling plan*. The MLO algorithm has an internal threshold parameter which needs to be hand-tuned in order for the optimization to succeed. This is contrary to the calibration free principle. During ARDEGO optimization, around three quarters of the evaluated designs violate resource constraints. The ARDEGO overhead time, is relatively insignificant, at around two hours.

III. Conclusions and Future Work

We present ARDEGO, a *calibration free* algorithm for automatic optimization of design parameters in reconfigurable designs. ARDEGO is evaluated using two case studies, a quadrature-based financial application and an RTM design for seismic imaging. The quadrature design shows a clear time saving when using ARDEGO over a manual approach, hill climbing and the MLO optimization algorithm. In the RTM design, we show that ARDEGO can optimize highly-dimensional designs. Although the ARDEGO algorithm can still take substantial amount of time to finish optimization, it offers a clear advantage over manual approaches that require extensive designer interaction to study and tweak the design. Current and future work includes further evaluation of the approach and acceleration of the most time-consuming parts of the algorithm.

Acknowledgement. This work is supported by the European Union Seventh FrameworkProgramme under grant agreement nuber 257906, 287804 and 318521, by UK ESPRC, by Maxeler University Programme, by HiPEAC NoE, by Altera, and by Xlinx.

References

[1] A. H. Tse et al., "Optimising performance of quadrature methods with reduced precisions," in *ARC*. Springer, 2012, pp. 251–263.

[2] Q. Jin et al., "Optimising explicit finite difference option pricing for dynamic constant reconfiguration," in *FPL*, 2012, pp. 165–172.

[3] X. Niu et al., "Exploiting run-time reconfiguration in stencil computation," in *FPL*, 2012, pp. 173–180.

[4] J. Janusevskis et al., "Expected improvements for the asynchronous parallel global optimization of expensive functions: Potentials and challenges," in *Learning and Intelligent Optimization*. Springer, 2012, pp. 413–418.

[5] M. Kurek, T. Becker, and W. Luk, "Parametric optimization of reconfigurable designs using machine learning," in *ARC'13*. Springer, 2013, pp. 134–145.

[6] C. Pilato el. al, "Speeding-up expensive evaluations in high-level synthesis using solution modeling and fitness inheritance," in *Computational Intelligence in Expensive Optimization Problems*. Springer, 2010, vol. 2, pp. 701–723.

[7] C. Rasmussen and C. Williams, *Gaussian Processes for Machine Learning*. MIT Press, 2006.

[8] A. Basudhar et al., "Constrained efficient global optimization with support vector machines," *Struct. Multidiscip. Optim.*, vol. 46, no. 2, pp. 201–221, Aug. 2012.

[9] N. Hansen, "The CMA Evolution Strategy: A Comparing Review," in *Towards a New Evolutionary Computation*. Springer, 2006, vol. 192, pp. 75–102.

[10] M. D. Mckay et al., "A comparison of three methods for selecting values of input variables in the analysis of output from a computer code," *Technometrics*, pp. 55–61, 2000.