Parametric Optimization of Reconfigurable Designs using Machine Learning

Maciej Kurek, Tobias Becker, and Wayne Luk

Department of Computing, Imperial College London

Abstract. This paper presents a novel technique that uses meta-heuristics and machine learning to automate the optimization of design parameters for reconfigurable designs. Traditionally, such an optimization involves manual application analysis as well as model and parameter space exploration tool creation. We develop a Machine Learning Optimizer (MLO) to automate this process. From a number of benchmark executions, we automatically derive the characteristics of the parameter space and create a surrogate fitness function through regression and classification. Based on this surrogate model, design parameters are optimized with meta-heuristics. We evaluate our approach using two case studies, showing that the number of benchmark evaluations can be reduced by up to 85% compared to previously performed manual optimization.

Keywords: optimization, surrogate modeling, PSO, GP, SVM, FPGA

1 Introduction

Field programmable gate arrays (FPGAs) allow designs that are customized to the requirement of the application. Reconfiguration is an additional benefit which allows the designer to modify designs at run time, potentially increasing performance and efficiency. Unfortunately, the optimization of reconfigurable designs often requires substantial effort from designers who have to analyze the application, create models and benchmarks and subsequently use them to optimize the design. This process often involves adjusting multiple design parameters such as numerical precision, degree of pipelining or number of cores. One could proceed with automated optimization based on an exhaustive search through design parameters which are derived from application benchmarks; however, this is unrealistic since benchmark evaluations involve bitstream generation and code execution which often takes hours of compute time.

It has been shown useful to construct surrogate models of fitness functions representing design quality for computationally expensive optimization problems in various fields [1–5]. As these models are orders of magnitude faster to evaluate than the actual fitness function, they can substantially accelerate optimization thus allowing for an automated approach. This is the motivation behind our development of the MLO tool which we apply to the problem of reconfigurable designs parameter optimization. In [6] we present initial concepts on optimizing parameter configuration of reconfigurable designs using surrogate models. We

now present the formalism and experimental evaluation for our approach. The contributions of this paper are:

- A mathematical characterization of the parameter space for reconfigurable designs as well as a definition of a fitness function based on application benchmarks (Section 3).
- Generic surrogate models to approximate the fitness function using regression and classification. We combine surrogate models with meta-heuristics to provide a new MLO algorithm for automated optimization of reconfigurable designs (Section 4).
- An evaluation of our MLO approach in two case studies: (a) execution time of a run-time reconfigurable software-defined radio with varied degree of parallelism, and (b) and a previously used [6] throughput of a quadrature based financial application with varied precision (Section 5).

2 Background

When developing reconfigurable applications, designers are often confronted with a very large parameter space. As a result the parameter space exploration can take an immense amount of time. A number of researchers approach the problem of high-cost fitness functions and large design spaces in various fields [1–5] by having fitness functions combined with fast-to-compute surrogate models provided by a Gaussian Process (GP) for decreasing evaluation time. However most current surrogate models only consist of a regressor and do not take into account possible invalid configurations within the design space. Furthermore, all of them are evaluated using artificial benchmarks spanned by continuous $\mathbb{R}^n$ spaces, while parameter spaces for reconfigurable applications usually also involve discrete dimensions (e.g. number of cores). Surrogate models approximating fitness functions by substituting lengthy evaluations with estimations based on closeness in a design space have been investigated in reconfigurable computing [7]. The work covers surrogate models for circuit synthesis from higher level languages (HLL), rather than parameter optimization.

GP is a machine learning technology based on strict theoretical fundamentals and Bayesian theory [8, 9]. GP does not require a predefined structure, can approximate arbitrary function landscapes including discontinuities, and includes a theoretical framework for obtaining the optimum hyper-parameters [4]. An advantage of GP is that it provides a predictive distribution, not a point estimate.

A Gaussian process is a collection of random variables, any finite set of which have a joint Gaussian distribution. A Gaussian process is completely specified by its mean function $m(\mathbf{x})$ and the covariance (kernel) function $k(\mathbf{x}, \mathbf{x}')$:

$$\hat{f}(\mathbf{x}) \sim \mathcal{GP}(m(\mathbf{x}), k(\mathbf{x}, \mathbf{x}')) \tag{1}$$

The $k(\mathbf{x}, \mathbf{x}')$ expresses the covariance between pairs of random variables, and in regression analysis it expresses the relation between input-output pairs. This

is based on a training set $\mathcal{D}$ of n observations, $\mathcal{D} = (\mathbf{x}_i, y_i)|i = 1, ...n$, where $\mathbf{x}$ denotes an input vector, y denotes a scalar output. The column vector inputs for all n cases are aggregated in the $D \times n$ design matrix X, and the outputs are collected in the vector $\mathbf{y}$. The goal of Bayesian forecasting is to compute the distribution $p(\hat{f}|\mathbf{x}_*, \mathbf{y}, X)$ of the function $\hat{f}$ at unseen input $\mathbf{x}_*$ given a set of training points $\mathcal{D}$. Using Bayes rule, the predictive posterior for the Gaussian process $\hat{f}$ and the predicted scalar outputs $\hat{f}(\mathbf{x}_*) = y_*$ can be obtained.

Support Vector Machine (SVM) is a maximum margin classifier, which constructs a hyperplane used for classification (or regression) [10]. SVMs use kernel functions $k(\mathbf{x}, \mathbf{x}')$ to transform the original feature space to a different space where a linear model is used for classification. SVMs are a class of decision machines and so do not provide posterior probabilities. There is a training set $\mathcal{D}$ of n observations, $\mathcal{D} = (\mathbf{x}_i, t_i)|i = 1, ...n$, where $\mathbf{x}$ denotes an input vector, t denotes a target value. The column vector inputs for all n cases are aggregated in the $D \times n$ design matrix X, and the targets in the vector $\mathbf{t}$. The goal is to classify an unseen input $\mathbf{x}_*$ based on X and $\mathbf{t}$ by computing a decision boundary.

Particle Swarm Optimization (PSO) is a population-based meta-heuristic based on the simulation of the social behavior of birds within a flock [11]. The algorithm starts by randomly initializing N particles where each individual is a point in the $\mathcal{X} = \mathbb{R} \times ... \times \mathbb{R}$ search space. The population is updated in an iterative manner where each particle is displaced based on its velocity v_{id}. The criteria for termination of the PSO algorithm can vary, and usually are determined by a time budget. The x_{id} represents the dth coordinate of particle i from the set X_* of N particles, where particle is a point within $\mathcal{X}$. In the most basic form of PSO Eq. 2-3 govern movement of particles. $r_1 \sim U(0, 1)$ and $r_2 \sim U(0, 1)$ are two independent uniformly distributed random numbers, c_1 and c_2 are acceleration coefficients and p_{gd} and p_{id} are dth coordinates of the global best and personal best positions. p_{gd} is updated when a new global best fitness is found and p_{id} is updated when a particle improves over its best fitness.

$$v_{id} = v_{id} + c_1 r_1 (p_{id} - x_{id}) + c_2 r_2 (p_{gd} - x_{id}) \tag{2}$$

$$x_{id} = x_{id} + v_{id} \tag{3}$$

3 Optimization Approach

Traditionally, optimization of reconfigurable applications is carried out by building benchmarks and relevant tools, and the associated analytical models [12, 13]. This involves the following steps:

1. Build application and a benchmark returning design quality metrics.
2. Specify search space boundaries and optimization goal.
3. Create analytical models for the design.

4. Create tools to explore the parameter space.
5. Use the tools to find optimal configurations, guided by the models in step 3.
6. If result is not satisfactory, redesign.

In our approach the user supplies a benchmark along with constraints and goals, and the MLO automatically carries out the optimization (Algorithm 1). Our approach consists of the following steps:

1. Build application and benchmark returning design quality metrics.
2. Specify search space boundaries and optimization goal.
3. Automatically optimize design with MLO.
4. If result is not satisfactory, redesign or revised time budget and search space.

Our idea of surrogate modeling is illustrated in Fig. 1. The MLO algorithm explores the parameter space by evaluating different benchmark configurations as presented in the left figure. The results obtained during evaluations are used to build a surrogate model which provides a regression of the fitness function and identifies invalid regions of the parameter space. A meta-heuristic (currently PSO) guides the exploration of the parameter space using the surrogate model.

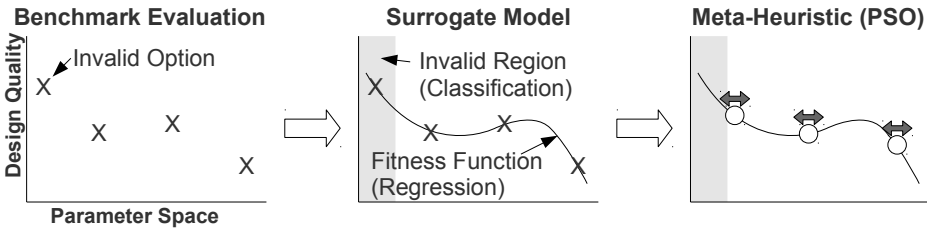

Fig. 1: Benchmark evaluations, surrogate model and model guided search.

3.1 Parameter Space

The parameter space $\mathcal{X}$ of a reconfigurable design is spanned by discrete and continuous parameters determining both the architecture and physical settings of FPGA designs. A vector $\mathbf{x}$ represents a parameter configuration within the parameter space $\mathcal{X} = \mathcal{X}_1 \times ... \times \mathcal{X}_D$ such that any $\mathcal{X}_d \subseteq \mathbb{R}$. If $\mathcal{X}_d \subseteq \mathbb{Z}$, its discretization level is independent of other dimensions. $\mathcal{X}_d$ can be bounded with upper and lower limits U_d, L_d such that for all x_d, $L_d \leq x_{id} \leq U_d$. An example of a continuous parameter is core frequency and an example of a discrete parameter is the number of compute cores. For all discrete dimensions the step size, which we define as smallest distance between any two x_{id}'s, can vary. We might only be able to increase memory width in 16 bits increments.

A discrete parameter space has important implications on the PSO algorithm, as the equations governing movements of particles Eq. 2-3 are defined for a

continuous $\mathbb{R}^n$ space. In Eq. 2 both r_1 and r_2 are random real numbers, which means that the resulting velocity component used to update position $\mathbf{x}$ cannot be used if x_d is discrete. To discretize the position value of a particle after its movement, we round its value and randomize its rounding error (dithering) presented in Eq. 4. By using dithering instead of truncation PSO particles maintain their velocity component which results in a more thorough exploration.

$$x_{di} = \begin{cases} \lfloor x_{id} \rfloor & U(0, stepsize) < (x_{di} \mod stepsize) \\ \lceil x_{id} \rceil & U(0, stepsize) \geq (x_{di} \mod stepsize) \end{cases} \tag{4}$$

3.2 Fitness Function

Given a parameter setting $\mathbf{x}$, the benchmark $b(\mathbf{x})$ returns a fitness metric which constitutes two values: y, the scalar metric of fitness and t, the exit code of the application. Execution time and power consumption are examples of fitness measures. There are be many possible exit codes t, with 0 indicating valid $\mathbf{x}$'s. The designer can choose to extend the benchmark to return additional exit codes depending on the failure cause, such as configurations producing inaccurate results or failing to build.

We distinguish three different types of exit codes. The first type is exit code 0 indicating a valid design. The second type of exit codes indicate configurations that produce results yet fail at least one constraint making them undesirable. The third type of exit codes is used for configurations that fail to produce any results. The region of $\mathcal{X}$ that defines configurations $\mathbf{x}$ that produce y and satisfy all constraints is defined as valid region $\mathcal{V}$, regions with designs failing at least one constraint yet producing y are part of failed region $\mathcal{F}$, and the region with designs failing to produce y is the invalid region $\mathcal{I}$. If $\mathbf{x}_*$ does not produce a valid result, we assign a value that the designer assumes to be the most disadvantageous. Depending on whether we face a minimization/maximization problem,s either a high max_{val} or low min_{val} value will be assigned.

$$f(\mathbf{x}) = \begin{cases} y & \mathbf{x} \in \mathcal{V} \\ max_{val} \vee min_{val} & otherwise \end{cases} \tag{5}$$

4 MLO Surrogate Model

We integrate a Bayesian regressor $\hat{f}$ and a classifier to create a novel surrogate model for a given fitness function f. As illustrated in Fig 1, the problem we face is regression of f over $\mathcal{V}$ and $\mathcal{F}$ as well as classification of $\mathcal{X}$. We make use of Bayesian regressors to access the probability of prediction of $\hat{f}(\mathbf{x}_*)$ of non-examined parameter configurations $\mathbf{x}_*$. We use classifiers to predict exit codes of X_* across $\mathcal{X}$. Regressions are made using the training set obtained from benchmark execution $\mathcal{D}_r$, while classification is done using the training set $\mathcal{D}_c$.

We invoke $regressor(\mathcal{D}_r, \mathbf{x}_*)$ for every particle in $\mathbf{x}_*$ to obtain the regression y_* and its probability $p(y_*|\mathbf{x}_*, \mathcal{D}_r)$, which we denote as ρ for simplicity. Class label t_* of particle $\mathbf{x}_*$ is predicted by the classifier $classifier(\mathcal{D}_c, \mathbf{x}_*)$.

Algorithm 1 MLO

for $\mathbf{x}_* \in X_*$ **do**
  $\mathbf{x}_*.fit \leftarrow f(\mathbf{x}_*)$ ▷ Initialize with a uniformly randomized set X_*.
end for
repeat
  for $\mathbf{x}_* \in X_*$ **do**
    $y_*, \rho \leftarrow regressor(\mathcal{D}_r, \mathbf{x}_*)$
    $t_* \leftarrow classifier(\mathcal{D}_c, \mathbf{x}_*)$
    if $\rho < min_\rho$ and $t_* = 0$ **then**
      $\mathbf{x}_*.fit \leftarrow y_*$
    else
      if $t_* = 0$ **then**
        $\mathbf{x}_*.fit \leftarrow f(\mathbf{x}_*)$
      else
        $\mathbf{x}_*.fit \leftarrow \max_{val}$ or $\min_{val}$
      end if
    end if
  end for
  $X_* \leftarrow Meta(X_*)$ ▷ Iteration of the meta-heuristic
until Termination Criteria Satisfied

We present our MLO in Algorithm 1. The algorithm's main novelty with respect to surrogate-based algorithms is the integration of a classifier to account for invalid regions of $\mathcal{X}$. We initialize the meta-heuristic of our choice with N particles X_* uniformly randomly scattered across $\mathcal{X}$. Each particle has an associated fitness $\mathbf{x}.fit$ and a position $\mathbf{x}$. For all $\mathbf{x}_*$ predicted to lie in $\mathcal{V}$ we proceed as follows. Whenever ρ returned by the regressor is smaller than the minimum required confidence min_ρ we use the y_*; otherwise we assume the prediction to be inaccurate and evaluate $f(\mathbf{x}_*)$. The meta-heuristic will avoid $\mathcal{I}$ and $\mathcal{F}$ regions as they are both assigned unfavorable max_{val} or min_{val} values. We construct the training sets $\mathcal{D}_c$ and $\mathcal{D}_r$ as described in Algorithm 2. Whenever $b(\mathbf{x}_*)$ is evaluated, $(\mathbf{x}_*, t_*)$ is included within the classifier training set $\mathcal{D}_c$. If exit code is valid ($t_* = 0$), then $(\mathbf{x}_*, y_*)$ is added to $\mathcal{D}_r$.

Although the MLO will converge towards an optimum, it is limited by heuristic search restrictions and as such it cannot guarantee to find the global optimum. Hence, it is crucial to specify the termination criteria. Determining MLO termination criteria is based on the optimization scenario and we present three possibilities where the user:

1. Has a limited compute time budget.
2. Requires only certain design quality.
3. Needs maximum performance, with a large budget.

A user can have a limited compute time budget when optimizing an application and the MLO can terminate once the budget has been reached. For example, we could allocate a number of machines for a 24 hour period. Alternatively, if the user only requires a certain performance, the MLO can be run until a configuration x is found that meets the required performance, and the optimization can be terminated. Lastly, if the MLO is used to maximize performance without a limited compute time budget, the MLO will terminate when the best found solution does not improve during a pre-defined amount of time.

Algorithm 2 $f(\mathbf{x})$

```
1: t, y ← b(x)
2: D_c ← (x, t)                          ▷ Update the classifier's training set
3: if t ∈ F or t ∈ V then
4:     D_r ← (x, y)                      ▷ Update the regressor's training set
5: end if
6: if t ∈ V then
7:     return y
8: else
9:     return max_val or min_val
10: end if
```

5 Evaluation

We use our approach to optimize two designs which are previously optimized with custom analytical models. The first application is a run-time reconfigurable software-defined radio with variable degree of parallelism [13]. The second is a quadrature-based financial application with variable precision [12], for which we show how known analytical models can be used to reduce the number of dimensions that need to be explored. We use GPs utilizing an anisotropic exponential kernel with additive Gaussian noise. We choose SVMs as our classifier with a Radial Basis Function (RBF) kernel. Due to its simplicity and effectiveness we use a velocity clamping version of PSO with c_1 and c_2 set to 2.0. All presented results are averaged over 20 trials. To evaluate the MLO performance in our three scenarios, we terminate when the global optimum is reached. We determine the globally optimal configuration with analytic methods, run the MLO to achieve the same value and then compare the complexity of both approaches.

As shown in Fig. 2 we create a surrogate model of the fitness function. We also classify the design space using SVM as shown in the right figure. We see regions of $\mathcal{X}$ with colour distinguishing different exit codes; dark gray for valid and light gray for inaccurate designs. Black x marks drawn over the design space represent configurations $\mathbf{x}$ which have been evaluated and used to build the surrogate model. The design space includes white circles which represent positions of the particles of the PSO algorithm during the iteration when the image was created.

5.1 Reconfigurable Software-defined Radio

We construct a benchmark based on studies conducted in [13]. The designer faces a trade-off between reconfiguration time and number of processing elements p. Larger values of p correspond to designs with higher compute throughput; however, the chip takes longer to reconfigure and our aim is to find the optimal value of p. The reconfigurable radio can run at two different chip reconfiguration bandwidths Φ_{config} of 5 MB/s or 300 MB/s.

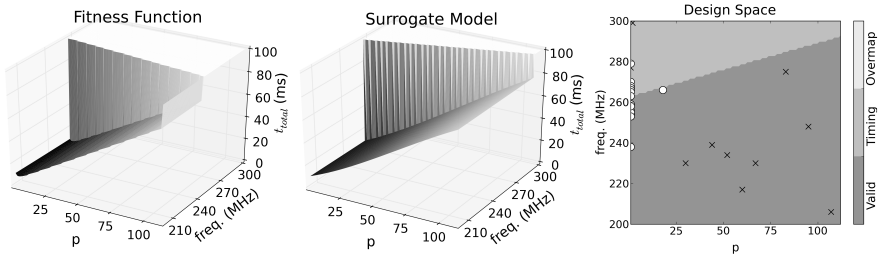

Fig. 2: Reconfigurable radio f ($\Phi_{config} = 5\,\text{MB/s}$) and its surrogate model.

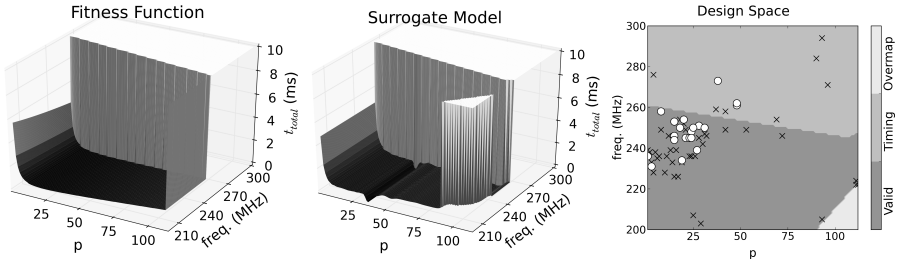

Fig. 3: Reconfigurable radio f ($\Phi_{config} = 300\,\text{MB/s}$) and its surrogate model.

Our parameters are p, Φ_{config} and core frequency $freq$. We define the design space as $\mathcal{X} = \{1 - 112\} \times \{5, 300\} \times \{freq_{min} - 300\}$. We change $\mathcal{X}$ by varying the minimum frequency $freq_{min}$. Although the problem is three-dimensional, due to low discretization level of Φ_{config} we treat it as two separate two-dimensional optimizations. For the $\mathcal{I}$ region constituting timing and FPGA resource overmapping regions, we mark the execution time as undesirable. MLO terminates when $\mathbf{x}$ is evaluated within 2 MHz range of globally optimal solution.

In Fig. 2 we see how the SVM classifies a fraction of the parameter space as $\mathcal{V}$ and how the surrogate model closely matches the fitness function. We also see how particles collapse and explore the optimal region $p \approx 4$ for $\Phi_{config} = 5\,\text{MB/s}$. In Fig. 3 we observe a similar situation but for $\Phi_{config} = 300\,\text{MB/s}$ with the

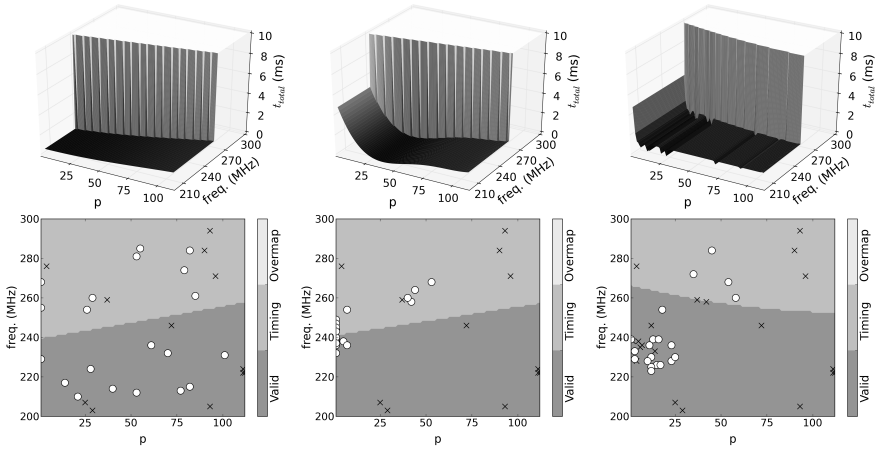

Fig. 4: Optimization of f ($\Phi_{config} = 300\,\mathrm{MB/s}$) after 13, 14 and 22 f evaluations.

optimal region $p \approx 20$. Again, the surrogate model resembles the fitness function. The collapse of particles is equivalent to the fine-tuning of the design parameters. We present a visualization of the optimization in Fig. 4, each pair of figures representing subsequent iterations. The top figures show the surrogate model, while the bottom figures represent corresponding visualization of the design space and its classification. During several initial iterations and f evaluations, the particles (shown as white circles) are misled by the surrogate model to explore $p \approx 4$ region. In the last figure we see particles guided by an improved surrogate model moving towards the optimum $p \approx 20$ region.

We use the reconfigurable radio benchmark to determine the impact of design space size on the convergence of the MLO algorithm. In Tab. 1 we see a tendency of the number of f evaluations to decrease along with the design space size. We trim design space by increasing the lower limit of admissible frequency $freq_{min}$. This shows that the designer should select a small parameter range as small design space improves MLO convergence. One outlier of $f = 54$ in the case of Φ_{config}=300 MB/s and $freq_{min}$=200 MHz can be explained with the overall small sample size. Manual optimization is replaced by MLO, which works with nearly no manual input but for the initial design space specification.

Table 1: Average number of f evaluations - Reconfigurable radio optimization.

Φ_{config}	$freq_{min}$ 150 MHz	200 MHz	220 MHz
5 MB/s	44	37	31
300 MB/s	47	54	45

5.2 Quadrature Method-based Application

In [12] the designer explores trade-off between accuracy and throughput in an application with three parameters. The first two parameters are mantissa width m_w of the floating point operators and the number of computational cores *cores*. Larger number of m_w bits increases computation accuracy, but limits the maximum number of *cores* that can be implemented on the chip due to the increased size of the individual core. The third parameter is the density factor d_f which specifies the density of quadratures used for integral estimation. It is a software parameter and is independent of the generated bitstream. Density factor d_f increases computation time per integration while improving the accuracy of the results due to finer estimation of the integral.

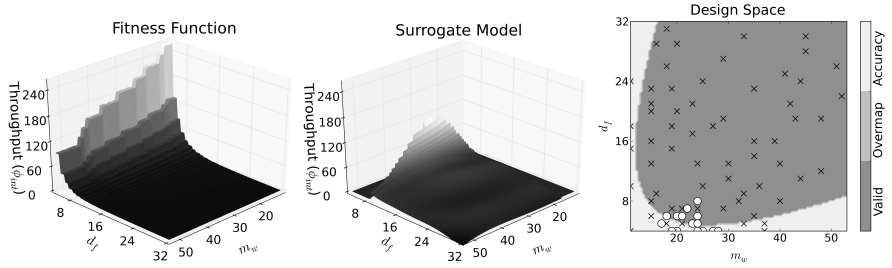

Fig. 5: Quadrature-based application f and its surrogate model.

The optimization goal is to find the design offering the highest throughput given a required minimum accuracy defined in terms of root mean square error ϵ_{rms}. The error is defined with respect to results obtained by calculating a set of reference integrals at the highest possible precision. The MLO terminates when the globally optimal configuration for a given ϵ_{rms} is found. The $\mathcal{F}$ region contains the inaccurate result class, as these benchmark evaluations can be reused for regression. The design space $\mathcal{X}$ is defined as $m_w \times cores \times d_f$: $\{11-53\} \times \{1-16\} \times \{4-32\}$. We can explore the whole $\mathcal{X}$ in a three-dimensional scheme or we can reduce the three-dimensional problem into two dimensions using an analytical resource usage estimation model. Resource usage is linearly related to *cores*, and after generating a single *core* bitstream we can create a simple analytical resource model which reduces the parameter space to two dimensions. Density factor d_f is a software parameter while m_w and *cores* affect the bitstream. Varying d_f only involves software execution, as long as a bitstream for the given m_w is already generated. If we evaluate a design with m_w (or m_w, *cores*) that has not been evaluated before, we generate a new bitstream.

We present a visualization of the two-dimensional optimization in Fig. 5, where the ϵ_{rms} limit is set to a value of 0.1. The bottom-left corner of $\mathcal{V}$ contains the global optimum which is difficult to determine without additional benchmark evaluations, as the maximum number of possible *cores* and therefore throughput

is limited by FPGA resources and as a result is chip dependent. Regions of space with low d_f or m_w are correctly predicted to offer low accuracy (light gray area).

Table 2: Average number of f evaluations - Quadrature application optimization.

cores	ϵ_{rms}	0.1	0.01	0.001
three-dimensional		138	67	47
two-dimensional		71	43	28

To measure the algorithms convergence the MLO terminates when the design with the highest throughput at the specified precision is found. The number of required f evaluations is shown in Tab. 2. The previously suggested optimization scheme [12] involves generating bitstreams for the full m_w range. Using our MLO combined with the analytical resource model, we reduce the number of bitstream generations as we avoid exploring *cores* and thus decrease the design space. Around 20-50% of f evaluations involve bitstream generation. The number has a high variance between individual runs as the swarm either skips undesirable configurations or thoroughly explore the whole design space.

The optimization scheme presented in [12] involves generating all possible bitstreams with $cores = 1$, and a binary search of the d_f values. Once the optimal (d_f, m_w) tuple is found, the number of *cores* can be determined. It also requires the generation of bitstreams for all m_w resulting in 53-11=42 distinct bitstreams. Furthermore, the number of bitstreams is nearly doubled since after the first generation, usually a second bitstream generation follows to adjust *cores*. Binary search is performed on the d_f range of 32-4=28 distinct values per bitstream, which yields on average $2 \times \lceil \log_2 (28) \rceil = 10$ benchmark evaluations per bitstream.

In comparison the MLO performance can be measured both in terms of f evaluations and bitstream generations. Using the optimization approach from [12] we perform a binary search on d_f range for all m_w values resulting on average $10 \times 42 = 420$ f evaluations regardless of the ϵ_{rms} limit. As presented in Tab. 2 for $\epsilon_{rms} = 0.1$ the MLO requires 75 evaluations (85 % less) in the two-dimensional scheme and 138 (67 % less) in the three-dimensional scheme. The number becomes more favorable for the MLO when ϵ_{rms} is reduced as $\mathcal{V}$ is decreased and the MLO needs to explore a smaller area, while average number of f evaluations in their optimization approach stays constant. Not all f evaluations involve bitstream generations: for $\epsilon_{rms} = 0.1$, 50% of f evaluations involve two bitstream generations resulting in 71 bitstreams compared to 82 bitstreams in [12]. In the three-dimensional scheme, MLO further decreases number of bitstream generations, to an average of 69. Our automated approach clearly outperforms manual design both in terms of f evaluations and bitstream generations, although in the second case the results are less dominant.

6 Conclusions and Future Work

We have proposed MLO, a novel tool which can determine optimized parameter configuration of a reconfigurable FPGA design. The MLO can offer superior performance, while reducing effort on analysis and application-specific tool development. The main advantage of using the MLO is a shift from manual optimization to automatic computation. The MLO requires multiple benchmarks for further evaluation, and there are many opportunities for future work; an example is the development of new surrogate models that would allow the reduction of required benchmark samples and efficiently address high dimensional examples. There are numerous cases where level of parallelism, timing and other parameters span tens of dimensions and would benefit from an effective automated approach.

Acknowledgements This work is supported by the European Union Seventh Framework Programme under grant agreement number 248976, 257906, 287804 and 318521, by UK EPSRC, by Maxeler University Programme, and by Xilinx.

References

1. Y. Jin, M. Olhofer, and B. Sendhoff, "A framework for evolutionary optimization with approximate fitness functions," *IEEE Transactions on Evolutionary Computation*, vol. 6, no. 5, pp. 481–494, 2002.
2. Y.S. Ong and et al., "Evolutionary optimization of computationally expensive problems via surrogate modeling," *AIAA*, vol. 41, no. 4, pp. 689–696, 2003.
3. G. Su, "Gaussian process assisted differential evolution algorithm for computationally expensive optimization problems," in *PACIIA*. IEEE Computer Society, 2008, pp. 272–276.
4. S. Guoshao and J. Quan, "A cooperative optimization algorithm based on gaussian process and particle swarm optimization for optimizing expensive problems," in *CSO*, vol. 2, 2009, pp. 929–933.
5. H.A.L. Thi, D.T. Pham, and N.V. Thoai, "Combination between global and local methods for solving an optimization problem over the efficient set," *EJOR*, vol. 142, no. 2, pp. 258–270, 2002.
6. M. Kurek and W. Luk, "Parametric Reconfigurable Designs with Machine Learning Optimizer," in *FPT*, 2012.
7. C. Pilato and et al., "Improving evolutionary exploration to area-time optimization of FPGA designs," *J. Syst. Archit.*, vol. 54, no. 11, pp. 1046–1057, 2008.
8. M. Seeger, "Gaussian processes for machine learning," *International Journal of Neural Systems*, vol. 14, pp. 69–106, 2004.
9. C. Rasmussen and C. Williams, *Gaussian Processes for Machine Learning*. MIT Press, 2006.
10. C. M. Bishop, *Pattern Recognition and Machine Learning*. Springer-Verlag, 2006.
11. F. Van Den Bergh, "An analysis of particle swarm optimizers," Ph.D. dissertation, University of Pretoria, South Africa, 2002.
12. A.H.T. Tse and et al., "Optimising performance of quadrature methods with reduced precisions," in *ARC*. Springer, 2012, pp. 251–263.
13. T. Becker, W. Luk, and P. Y. Cheung, "Parametric design for reconfigurable software-defined radio," in *ARC*. Springer, 2009, pp. 15–26.