Parametric Reconfigurable Designs with Machine Learning Optimizer

Maciej Kurek, Wayne Luk

Department of Computing, Imperial College London

***Abstract*—We investigate the use of meta-heuristics and machine learning to automate reconfigurable application parameter optimization. The traditional approach involves two steps: (a) analyzing the application in order to create models and tools for exploration of the parameter space, and (b) exploring the parameter space using such tools. The proposed approach, called the Machine Learning Optimizer (MLO), involves a Particle Swarm Optimization (PSO) algorithm with an underlying surrogate fitness function model based on Gaussian Process (GP) and Support Vector Machines (SVMs). We present a case study of a quadrature based financial application with varied precision. We evaluate our approach by comparing the amount of benchmark evaluations and bit-stream generations when using MLO and when using the traditional approach.**

I. Introduction

The optimization of heterogeneous computing applications often requires substantial effort from the designer who has to analyze the application, create models and benchmarks and subsequently use them to optimize the application. One could try to employ exhaustive search of the application parameter space to carry out optimization yet it is unrealistic since benchmark evaluations involve bit-stream generation and code execution which takes hours of computing time. Recently it has been shown useful to use surrogate models combined with fitness functions for computationally expensive optimization problems in various fields [1], [2], [3], [4], [5]. As these models are orders of magnitude cheaper, they can substantially decrease optimization cost thus allowing for an automated approach. This is the motivation behind MLO which we apply to non-linear and multi-modal problem of heterogeneous application parameter optimization. We use GPs to model performance of the design like execution time or throughput, while searching for the global optimum using PSO. We classify the parameter space using SVMs to identify designs that would fail constraints; over-map on resources, produce inaccurate results or other. Our contributions:

- The proposed approach has two new aspects: (a) surrogate models of application benchmarks (b) automated application optimization scheme based on the surrogate models and MLO (Section II).
- We use our approach to optimize throughput of a quadrature based financial application with varied precision [6]. We show how MLO can significantly decrease benchmark evaluations (up to 85%) and can be used in combination with analytical models (Section III).

978-1-4673-2845-6/12/$31.00 © 2012 IEEE

A. Gaussian Process Regression

GP is a newly developed machine learning technology based on strict theoretical fundamentals and Bayesian theory [7], [8]. GP does not require a predefined structure, can approximate arbitrary function landscapes including discontinuities and multi-modality, can have meaningful hyper-parameters, and includes a theoretical framework for obtaining the optimum hyper-parameters [4]. A further 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 $m(x)$ and the covariance (kernel) function $k(x, x')$:

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

The covariance function express the covariance between pairs of random variables, in case of regression they express our belief on relation between input, output pairs. There is a training set $\mathcal{D}$ of n observations, $\mathcal{D} = (x_i, y_i)|i = 1, ...n$, where x denotes an input vector, y denotes a scalar output or target. The column vector inputs for all n cases are aggregated in the $D \times n$ design matrix X, and the targets are collected in the vector y. The goal of Bayesian forecasting is to compute the distribution $p(x_*, y_*, \mathcal{D})$ of output y_* given a test input x_* and a set of training points $\mathcal{D}$. Using Bayes rule, the posterior distribution for the Gaussian process outputs y_* can be obtained. By conditioning on the observed targets in the training set, the predictive distribution is Gaussian.

B. Support Vector Machines Classification

The SVM is a maximum margin classifier, which constructs a hyperplane used for classification (or regression) [9]. SVMs use kernel functions $k(x, x')$ to transom 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, $\mathcal{D} = (x_i, t_i)|i = 1, ...n$, where 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 are collected in the vector t. The goal is to compute a decision boundary that will allow to categorize inputs x_* based on X and t.

C. Particle Swarm Algorithm (PSO)

PSO is a population-based meta-heuristic based on the simulation of the social behavior of birds within a flock. The algorithm starts by randomly initializing N particles where each individual is a point in the $\mathcal{X}$ search space with inertia governing its movement. The population is updated in an iterative manner where each particle is displaced using its velocity. The criteria for termination of the PSO algorithm can vary, and usually is determined by time budget. There are a large number of variations of the PSO algorithm, dealing with different types of search space, different search space boundary restrictions - as well as adaptive flavors that automatically adjust algorithms parameters. The x_{di} represents the dth coordinate of particle i from the set X_* of N particles, where particle is a point within $\mathcal{X}$. Eq. 2-3 govern movement of a particle. v_{di} is the velocity of particle, where r_1 and r_2 are two independent uniformly distributed random numbers, c_1 and c_2 are constants, w is acceleration factor and p_{gd} and p_{di} are global best and particle best positions.

$$v_{di} = wv_{di} + c_1 r_1(p_{di} - x_{di}) + c_2 r_2(p_{gd} - x_{di}) \quad (2)$$

$$x_{di} = x_{di} + v_{di} \quad (3)$$

II. Design

Traditionally optimization is carried out by building benchmarks and relevant tools, often constructing analytical models and finally evaluating a set of parameter configurations:

1) Build benchmark returning design quality metrics.
2) Specify search space boundaries and optimization goal (maximization/minimization).
3) Create analytical models of the design.
4) Create tools used to explore the parameter space.
5) Use the tools to find optimal configurations, guided by the models in step 3.
6) If result not satisfactory redesign.

In our approach the user supplies a benchmark along with the constraints and goals, and using meta-heuristics MLO automatically carries out the optimization (Algorithm 1). We present our approach to parameter optimization:

1) Build benchmark returning design quality metrics.
2) Specify search space boundaries and optimization goal (maximization/minimization).
3) Provide MLO (Algorithm 1) with time budget and use it to optimize the application.
4) If result not satisfactory redesign or increase budget/search space.

Our steps 1, 2 and 4 correspond to traditional steps 1, 2 and 6. Therefore the MLO approach will be faster than the traditional approach when the time for step 3 of the MLO approach is shorter than the total time for steps 3, 4 and 5 of the traditional approach.

We use meta-heuristics to explore parameter space. The challenge with applying meta-heuristics to reconfigurable application parameter optimization is related to high cost of fitness function evaluations (application benchmark evaluations). Our proposed optimization approach has two new aspects that counter this problem. We build a surrogate model of application fitness function (Section II-B) and we use it to define the new surrogate model aided meta-heuristic MLO (Section II-C). This allows us to substantially decrease amount of bit-stream generation and benchmark evaluations.

A. Parameter space and Fitness Function

The parameter space $\mathcal{X}$ of a heterogeneous application is spanned by discrete and continuous parameters determining both the architecture and physical settings of field programmed gate array (FPGA) designs. When provided a parameter setting x benchmark returns fitness $f(x) = (y(x), t(x))$ which constitutes of two values, a scalar metric of fitness $y(x)$ and the exit code of the application $t(x)$. $y(x)$ represents a metric provided by execution of the applications benchmark, like execution time or power consumption. There are can be many possible exit codes $t(x)$, 0 usually indicating valid $x's$. The designer can choose to extend benchmark to return extra exit codes depending on failure cause. A separate exit code can exist for configurations that produce in-accurate results, while a different ones for one which fail to build a bit-stream.

B. MLO Surrogate Model

We integrate Bayesian regressors and a classifier to create a novel and more powerful surrogate model for the defined f. The problem we face is regression of y over regions of $\mathcal{X}$ that produce valid results. We make use of Bayesian regressors to access the probability of prediction of $y(x)$. We use classifiers to predict exit codes of non-examined parameter configurations across $\mathcal{X}$. Regressions are made using training set $\mathcal{D}_{regressor}$, while classification is done using training set $\mathcal{D}_{classifier}$. When doing a prediction $y_*, p(x_*, y_*, \mathcal{D}_{regressor}) \leftarrow$ Regressor($\mathcal{D}_{regressor}, x_*$). We denote $p(x_*, y_*, \mathcal{D}_{regressor})$ as ρ for simplicity. $t_* \leftarrow$ Classifier($\mathcal{D}_{classifier}, x_*$), t_* is the predicted class of x_*.

C. MLO Algorithm

We sketch MLO in Algorithm 1. The algorithms main new aspect with respect to previously mentioned surrogate based algorithms; 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 associated fitness $x_*.fit$ and a position x_*. Whenever point x_* is evaluated $(x_*, t(x_*))$ is included within the classifier training set $\mathcal{D}_{classifier}$. If $f(x_*)$ produces $y(x_*)$ and $t_* = 0$, $(x_*, y(x_*))$ is added to $\mathcal{D}_{regressor}$.

For all x_* in predicted $t(x_*) = 0$ we proceed as follows; Whenever ρ returned by regressors is larger then max_ρ we use the y_*, otherwise we believe prediction to be inaccurate and evaluate $f(x_*)$. x_* in all regions that do not produce valid results are assigned an arbitrary large/small value (min_{val}/max_{val}) marking them as unwanted (depending on whether we face a minimization/maximization problem) and assumed to be accurately modeled. Meta-heuristic will avoid invalid regions as they are assigned unfavorable values.

Algorithm 1 MLO

```
procedure MLO
    for x_* ∈ X_* do   ▷ Initialize with a uniformly randomized set X_*.
        f(x_*)
    end for
    repeat
        for x_* ∈ X_* do
            y_*, ρ ← Regressor(D_regressor, x_*)
            t_* ← Classifier(D_classifier, x_*)
            if ρ < max_ρ and t_* = 0 then
                x_*.fit ← f(x_*)
            else
                if t_* = 0 then
                    x_i.fit ← y_*
                else
                    x_i.fit ← max_val or min_val
                end if
            end if
        end for
        X_* ← Meta(X_*)   ▷ Iteration of the meta-heuristic
    until Iteration/Fitness budget exceeded
end procedure
```

III. Evaluation

We use our approach to optimize throughput of quadrature method-based financial application with varied precision [6] implemented on the Maxeler platform. We show how analytical models can aid MLO. We use a velocity clamping version of PSO algorithm as our meta-heuristic. A wide variety of regressors could be used to construct a surrogate model of y, yet there are some open limitations which can be hard to overcome [7], [8], [10]. The GP should be powerful enough to model the heterogeneous application, as numerous multi-modal and multi-dimensional non-linear functions were used as benchmarks for meta-heuristics along with GP based surrogate models [1], [2], [3], [4], [5]. We choose SVMs as our classifier. Different kernel functions can be used to express different beliefs on the shaped of boundaries and classes. We use anisotropic exponential kernel function with additive Gaussian noise kernel for the GP and a Radial Basis Function (RBF) kernel for the SVM.

In [6] an optimization model (traditional step 3) is proposed along with an associated algorithm (traditional step 4) that improves design throughput (integrations per second ϕ_{int}) by varying the number representation in quadrature based applications. The application is parameterized with density factor d_f of integral estimation and the number mantissa width m_w. d_f is software parameter while m_w and $cores$ affects the bit-stream. If we evaluate a design with m_w (or m_w, $cores$) that has not been evaluated before, we generate a new bit-stream. Varying d_f only involves software execution, as long as the bit-stream has already been evaluated. As a reference design we use $m_w = 53, d_f = 32$ and fix FPGA frequency to 100 MHz, allowing us to meet timings (In [6] $d_f = 20$ is used to produce reference values). We follow with either a two dimensional optimization over m_w and d_f, or a three dimensional including $cores$. The two-dimensional scheme represent a combined approach where analytical model $cores = \lfloor 1/U_{slice} \rfloor$ is used to estimate the maximum number of cores based on bit-stream slice utilization U_{slice}. We generate a single core bit-stream and depending on U_{slice} (and m_w indirectly) generate a second multi-core bit-stream. In the three-dimensional case whole $\mathcal{X}$ is explored and modeled. According to our definition benchmark design is step 1 in both approaches.

$\mathcal{X}$ is defined as $\{11-53\}\times\{4-32\}\times\{1-16\}$, the parameter names are m_w, d_f and $cores$. The application terminates when the optimizer evaluates the design with highest throughput (maximization) at specified precision (ϵ_{rms}), allowing us to determine the convergence rate of MLO. We evaluate a single benchmark portfolio rather then a range [6]. Determining the acceptable parameter range is step 2 in both approaches.

We run MLO (MLO step 3) and present results in Tab. I. Around 20-50% of benchmark evaluations involve new bit-stream generation. The suggested optimization scheme [6] (traditional step 5) involves generating bit-streams for full m_w range, using MLO combined with analytical approach we decrease the number of bit-stream generations as we avoid a range of m_w. As an example time for step 3 of the MLO approach and time for step 5 in traditional approach can be both measured in terms f or benchmark evaluations. In case of $\epsilon_{rms} = 0.1$, we can see in in Tab. I that even if 50% of f evaluations involve bit-stream generations we end up with total of 50% $\times$ 71 = 35 vs. 42, as we limit m_w to 11-53 bits resulting in 42 different m_w values [6]. As of benchmark evaluations, number is still favorable for MLO. Compared to binary search [6] on a 28 d_f range on average 10 ($\lceil \log_2(28) \rceil$) benchmarks with different d_f have to be evaluated per m_w resulting in $10\times42 = 420$ f vs. 71 f (85 % less). Performance is also favorable for three-dimensional optimization, where MLO requires 138 f evaluations.

Fig. 1. represents throughput $y(x)$ spanned across all possible parameters settings $\mathcal{X}$. Fig. 2-4 visualizing state of MLO include; regions of $\mathcal{X}$ classified by SVM with colors distinguishing different exit codes; dark for valid and light gray for inaccurate designs; we do not encounter overmapping. Black x marks represent x which have been evaluated and included in $\mathcal{D}$, the white dots represent particles during the given iteration. The right image shows surrogate model spanned across $\mathcal{X}$, its gray-scale is not synchronized with Fig. 1. The optimization will start with random particle set up as presented in Fig. 2 where we see an initial evaluation of the model dependent on d_f and m_w. As the ϵ_{rms} limit is high, we can see SVM classifying whole of $\mathcal{X}$ as valid. After a number of PSO iterations and extra 25 f evaluations later, shown in Fig. 3, particles are moving towards optimum in the left bottom corner as indicated by black x trail behind the particles (white dots). Regions of space with low d_f or m_w are correctly predicted to offer low accuracy (light gray area) and the model shows similar shape to Fig. 1 being good indicator of accuracy. When we increase the ϵ_{rms} limit learned valid region decreases as smaller fraction of settings offer accurate results as seen in Fig. 4. We see particles exploiting corner of the valid region, as should always be the case [6]. The corner of valid region is difficult to determine without benchmark evaluations, as

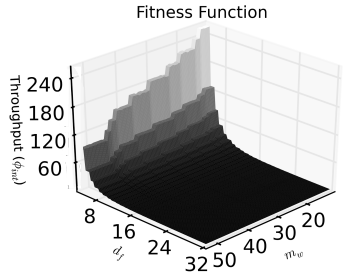

Fig. 1: Throughput $y(x)$ spanned across $\mathcal{X}$

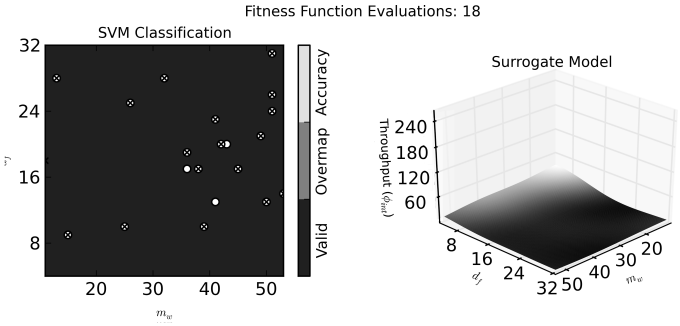

Fig. 2: MLO state after 18 f evaluations. (ϵ_{rms} = 0.1)

maximum number of possible $cores$ and therefore throughput is limited by FPGA resources and as a result chip dependent. Images we present are similar to ones presented in [6].

As shown before, the proposed MLO approach involves fewer bit stream generations when compared with the traditional approach, up to a reduction of 85% in the case of a financial benchmark based on the quadrature algorithm. Moreover, since our MLO approach adopts a generic algorithm (Algorithm 1), it often takes significantly less time to use it for a given application, than the traditional approach which would involve creating multiple application-specific algorithms for parameter optimization, one for each application.

IV. Future Work and Conclusions

We have shown how MLO can be used to determine optimal parameter configuration of application and how its settings affect MLO performance. We have also shown how analytical models can improve MLO performance. MLO can offer superior performance, save time on analysis and application specific tool development. Being far from mature, MLO requires multiple benchmarks for further evaluation. Although showing much promise, there are many opportunities for further work such as multi-objective optimization: finding acceptable performance and power consumption while maximising energy efficiency.

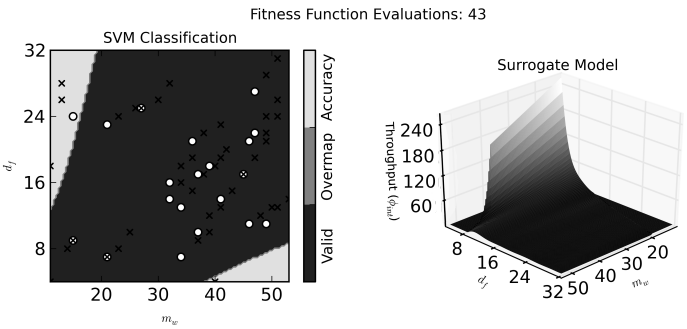

Fig. 3: MLO state 25 f evaluations later. (ϵ_{rms} = 0.1)

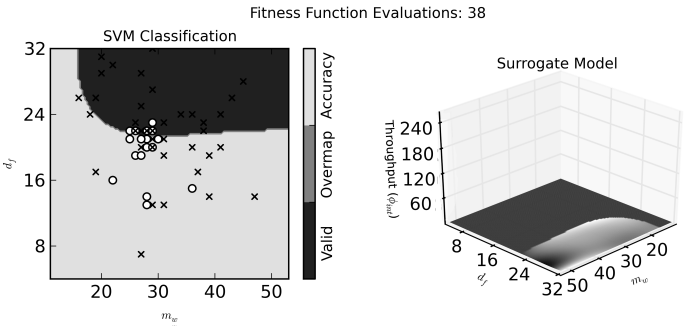

Fig. 4: MLO state after 38 f evaluations. (ϵ_{rms} = 0.01)

TABLE I: f evaluations - Quadrature application optimization.

$cores$	ϵ_{rms}	0.1	0.01	0.001
without		138	67	47
with		71	43	28

Acknowledgement. The research leading to these results has received funding from the European Union Seventh Framework Programme under grant agreement no. 257906. The support of UK EPSRC, Xilinx, Maxeler Technologies and Python community is gratefully acknowledged.

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, Oct. 2002.

[2] Y.S. Ong, P.B. Nair, and A.J. Keane, "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." IEEE PACIIA, 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, April 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] A.H.T. Tse, G.C.T. Chow, Q. Jin, D.B. Thomas, and W. Luk, "Optimising performance of quadrature methods with reduced precisions," in *In International Symposium on Applied Reconfigurable Computing (ARC)*, 2012, pp. 251–263.

[7] M. Seeger, "Gaussian processes for machine learning," *International Journal of Neural Systems*, vol. 14, pp. 69–106, 2004.

[8] C. Rasmussen and C. Williams, *Gaussian Processes for Machine Learning*. Cambridge, Massachusetts, USA: Mit Press, 2006.

[9] C. M. Bishop, *Pattern Recognition and Machine Learning*. Secaucus, NJ, USA: Springer-Verlag New York, Inc., 2006.

[10] D. Buche, N. Schraudolph, and P. Koumoutsakos, "Accelerating evolutionary algorithms with gaussian process fitness function models," *IEEE TSMC*, vol. 35, no. 2, pp. 183–194, May 2005.