Imperial College London
Department of Computing

Automated Optimization of Reconfigurable Designs

Maciej Kurek

Submitted in part fulfillment of the requirements for the degree of Doctor of Philosophy in Computing of the Imperial College London and the Diploma of Imperial College London, September 2015

Declaration of Originality

The work presented in this thesis is the result of my own original research. Wherever possible, references to the literature are provided and collaborative work is acknowledged.

Abstract

Currently, the optimization of reconfigurable design parameters is typically done manually and often involves substantial amount effort. The main focus of this thesis is to reduce this effort. The designer can focus on the implementation and design correctness, leaving the tools to carry out optimization. To address this, this thesis makes three main contributions.

First, we present initial investigation of reconfigurable design optimization with the Machine Learning Optimizer (MLO) algorithm. The algorithm is based on surrogate model technology and particle swarm optimization. By using surrogate models the long hardware generation time is mitigated and automatic optimization is possible. For the first time, to the best of our knowledge, we show how those models can both predict when hardware generation will fail and how well will the design perform.

Second, we introduce a new algorithm called Automatic Reconfigurable Design Efficient Global Optimization (ARDEGO), which is based on the Efficient Global Optimization (EGO) algorithm. Compared to MLO, it supports parallelism and uses a simpler optimization loop. As the ARDEGO algorithm uses multiple optimization compute nodes, its optimization speed is greatly improved relative to MLO. Hardware generation time is random in nature, two similar configurations can take vastly different amount of time to generate making parallelization complicated. The novelty is efficient use of the optimization compute nodes achieved through extension of the asynchronous parallel EGO algorithm to constrained problems.

Third, we show how results of design synthesis and benchmarking can be reused when a design is ported to a different platform or when its code is revised. This is achieved through the new Auto-Transfer algorithm. A methodology to make the best use of available synthesis and benchmarking results is a novel contribution to design automation of reconfigurable systems.

Acknowledgements

The first person I would like to thank is my supervisor Professor Wayne Luk. He has been a mentor and a friend to me since the final year of my undergraduate studies. I am most grateful for the trust and freedom he granted me. He always encouraged me to work on things I believe in, which others would consider risky or unconventional.

Special thanks to Tobias Becker, Sanjay Bilakhia, Thomas Chau, Giulia Ferretti, Xinyu Niu, Timothy Todman and Robert Wolstenholme for the help and advise they offered. The biggest challenge I faced during my research was to bring together the Bayesian optimization and reconfigurable computing worlds. All of them have helped me on numerous occasions by discussing my work, reviewing my papers and providing suggestions on how to better explain different aspects of work.

Special thanks to Stephen Weston. I have worked with Stephen during my internship at JPMorgan Chase during my first year of the PhD program, as well as at Maxeler Technologies during my write up period. Skills I learned during that time proved to be invaluable during my work on this Thesis. Furthermore, I would like to thank Steve Hutt and Florian Widmann, both of whom were my managers while working at Maxeler Technologies during my write up period. In particular, Steve has helped me develop a better, more thorough and precise approach to work.

I would like to thank Marc Deisenroth for his feedback on my late stage report and the help he has offered afterwards. He has been of great help in structuring the thesis and making the Bayesian Optimization components more approachable. His questions provided invaluable feedback.

Lastly, I would like to thank everyone in the Department of Computing at Imperial College London. That includes James Arram, Brahim Betkaoui, Pavel Burovskiy, Bridgette Cooper, Kit Cheung, Gabriel De Figueiredo Coutinho, Stewart Denholm, Paul Grigoras, Ce Guo, Liucheng Guo, Eddie Hung, Gordon Inggs, Qiwei Jin, Adrien Le Masle, Nicholas Ng, Shengjia Shao, David Thomas, Anson Tse, Shulin Yan, Jinzhe Yang. Hopefully I was not too much of an annoyance building countless numbers of bitstreams and constantly occupying our CAD machines.

Dedications

To my parents,
For teaching me the value of hard work and all the support you have given me throughout all of those years, even if at times I did not seem as grateful as I should.

Publications

- M. Kurek and W. Luk, "Parametric Reconfigurable Designs with Machine Learning Optimizer", *International Conference on Field-Programmable Technology (FPT)*, 2012.

- M. Kurek, T. Becker and W. Luk, "Parametric Optimization of Reconfigurable Designs Using Machine Learning", *International Symposium on Applied Reconfigurable Computing (ARC)*, 2013.

- M. Kurek, T. Liu and W. Luk, "Multi-objective Self-optimization of Reconfigurable Designs with Machine Learning", *Self-Awareness in Reconfigurable Computing Systems (SRCS)*, 2013.

- M. Kurek, T. Becker, T.C.P. Chau and W. Luk, "Automating Optimization of Reconfigurable Designs", *International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, 2014.

- T.C.P. Chau, M. Kurek, James S. Targett, J. Humphrey, G. Skouroupathis, A. Eele, J. Maciejowski, B. Cope, K. Cobden, P. Leong, P. Y.K. Cheung and W. Luk, "SMCGen: Generating Reconfigurable Design for Sequential Monte Carlo Applications", *International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, 2014.

Contents

Acronyms

ACO Ant Colony Optimization. 65

ALU Arithmetic Logical Unit. 48

ANN Artificial Neural Network. 67

API Application Programming Interface. 51, 52, 111, 195

ARD Automatic Relevance Determination. 76, 103, 112, 115, 119, 121, 122, 125–127, 140

ARDEGO Automatic Reconfigurable Design Efficient Global Optimization. 7, 32, 33, 35, 36, 44, 45, 129–131, 134–138, 141, 143–145, 148–158, 162–167, 172, 174, 175, 181, 184, 192, 193, 196, 197, 199

ASIC Application Specific Integrated Circuits. 35, 38, 47–50

BRAM Block RAM. 34, 48, 159, 190, 191, 199

CAD Computer Aided Design. 40, 41, 54, 80, 81, 130, 165

CAM Content Addressable Memory. 48

CPLD Complex Programmable Logic Device. 38, 48

CPU Central Processing Unit. 43, 50, 51, 151, 161, 181

DSU Debug Support Unit. 48

EGO Efficient Global Optimization. 7, 29, 32, 45, 85–88, 90, 91, 130, 136–138, 142, 143, 145, 196, 197

EI Expected Improvement. 85, 86, 89–91, 99, 137, 142–147, 150, 157, 192, 197

FIR Finite Impulse Response. 39, 42

FPGA Field Programmable Gate Array. 27, 28, 35, 37–39, 43, 47–52, 55, 57–60, 62, 64, 65, 91, 93, 94, 109–111, 126, 129, 148, 149, 153, 158–163, 177, 181–184, 186, 187

GA Genetic Algorithm. 65

GbSA Galaxy-based Search Algorithm. 65

GP Gaussian Process. 28, 29, 34, 73–79, 82–85, 89, 90, 94, 98–100, 102, 103, 105, 109, 111, 124, 135–137, 139, 140, 142, 143, 145–147, 151, 162, 163, 174, 181, 197, 199

GP-UCB Gaussian Process – Upper Confidence Bound. 84, 90, 198

GPU Graphic Processing Unit. 52

HDL Hardware Description Language. 27, 38, 49, 52

HLL Higher Level Language. 38, 51, 52

HLS High Level Synthesis. 52, 195

HPC High Performance Computing. 50

HSM Hybrid Surrogate Model. 83

IP Intellectual Property. 49

IWDP Intelligent Water Drops. 65

LUT Lookup Table. 34, 48, 49, 110, 149, 176, 182, 190, 191

MLO Machine Learning Optimizer. 7, 29–31, 35, 44, 45, 93–95, 97, 98, 100–102, 104, 105, 107–112, 115, 118–120, 122, 124–126, 129, 135, 148, 150, 152, 153, 155–157, 162, 163, 195–197, 199

MOMLO Multiobjective Machine Learning Optimizer. 34, 199, 200

MPI Most Probable Improvement. 90

PAR Place and Route. 31, 39, 58, 61–63, 110, 117, 130, 155, 163, 182, 190, 201

PES Predictive Entropy Search. 90

PQ Proximity Query. 28, 31, 32, 34, 39, 57, 58, 95, 109, 110, 117, 118, 131, 148, 149, 153, 155, 163, 182, 198, 200

PSO Particle Swarm Optimization. 30, 65, 83, 93, 97–100, 104–107, 109, 124, 136, 143, 144, 147, 148

RBF Radial Basis Function. 83

RTM Reverse Time Migration. 32, 34, 131, 148, 149, 156, 157, 162–164, 167, 181, 182, 186, 187, 192

RVM Relevance Vector Machine. 100, 137, 143, 199

SA Simulated Annealing. 65

SMC Sequential Monte Carlo. 95, 119, 181, 187, 188

SVM Support Vector Machine. 28, 66, 69–72, 79, 82, 91, 97–100, 102, 103, 107, 109, 111, 112, 117, 136, 137, 139, 140, 142, 143, 146, 147, 151, 155, 162, 163, 181, 198, 201

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation

The history of computing dates back to the ancient analog computers used for astronomical calculations. Yet, it was only within the last 100 years that many major advancements in physics, chemistry and mathematics allowed for the creation of modern digital computers. One of the major milestones was the development of general-purpose programming languages like C or Fortran in the 1960s-1970s. They offered a layer of abstraction from the underlying hardware allowing for faster software development and thus massively increased complexity of the programs. In particular, development of JAVA in the 1990s went as far as automating memory management [11]. Languages like Ruby or Python went even further, they were designed to allow the programmer to express complex functionality in just a few lines of code. The process of hardware abstraction in software design became so advanced, that currently many programmers have little to no understanding of the underlying computer architecture. The job of machine code generation has been offset to compilers. In the meantime, although not as widely acknowledged, the process of hardware design was also changing. Especially interesting was the development of reconfigurable hardware and logic synthesizers.

The initial idea of reconfigurable hardware is widely agreed to first appear in the 1960s [61]. It started gaining popularity in the 1990s with the increased availability of new FPGAs devices. Initially those devices could not compete with Application Specific Integrated Circuits (ASIC) both due to the performance and cost issues. However, in the early 2000s they became commercially viable due to the creation of better supporting software. Their main advantage over ASIC was lower up front development cost. When compared to Central Processing Units (CPUs) they offered better power utilization and

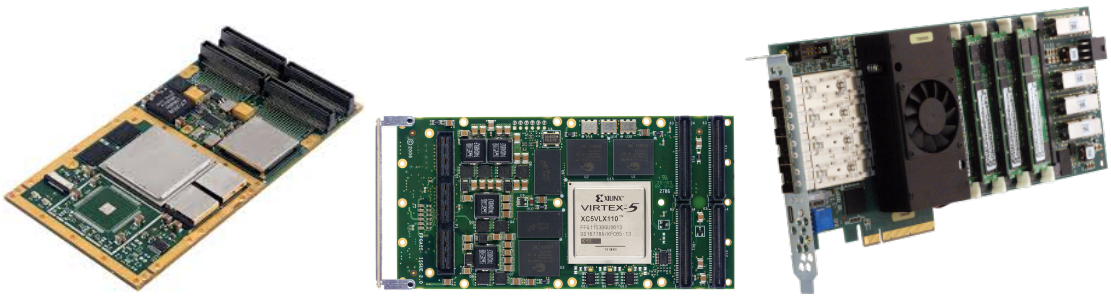


Figure 1.1: Some of many possible Field Programmable Gate Array (FPGA) boards: XMC-FPGA05D XMC/PM and PMC-FPGA05 with Xilinx Virtex-5 FPGAs from Curtiss Wright [1] and a Nallatech 395 with Atera Stratix V FPGA [2].

often orders of magnitude of speed-up.

While the technology has advanced, the reconfigurable hardware industry still faces a number of challenges. The spatial programming model used in FPGAs is very different from the widely understood temporal software programming model. Extracting spatial features from a temporal application written in C or Fortran and translating it to Hardware Description Language (HDL) consumes a large amount of engineering hours, often making FPGA accelerators economically infeasible. A number of ideas on automating the design process have been presented by various researchers, and some have been verified: new Higher Level Language (HLL), extensions of the existing languages or language specific accelerator libraries. In recent years the FPGA industry has been looking for the holy grail — an approach that would allow users to automatically and efficiently map HLL onto reconfigurable heterogeneous platforms. Such an approach would allow user to concentrate on developing an optimal algorithm, leaving most of the implementation to the system.

We are interested in providing domain experts like scientists and engineers with the power of reconfigurable computing while hiding the complexity of the application development. This scope being very broad, we focus on automatic optimization of reconfigurable design parameters. A good example of such a design is the Finite Impulse Response (FIR) filter

$$f(\mathbf{x}) = \sum_{i=0}^{q} \mathbb{b}_i \times \mathbf{x}_{q-i} \tag{1.1}$$

with a batch of q elements $\mathbf{x} = \{x_i\}_{i=1}^{q}$, where $x_i \in \mathbb{R}$, being processed using q coefficients $\mathbb{b}_i \in \mathbb{R}$ at every time step. Ideally multiple filters should be implemented in parallel, and the numerical representation of real numbers has to be specified. The design multiplication

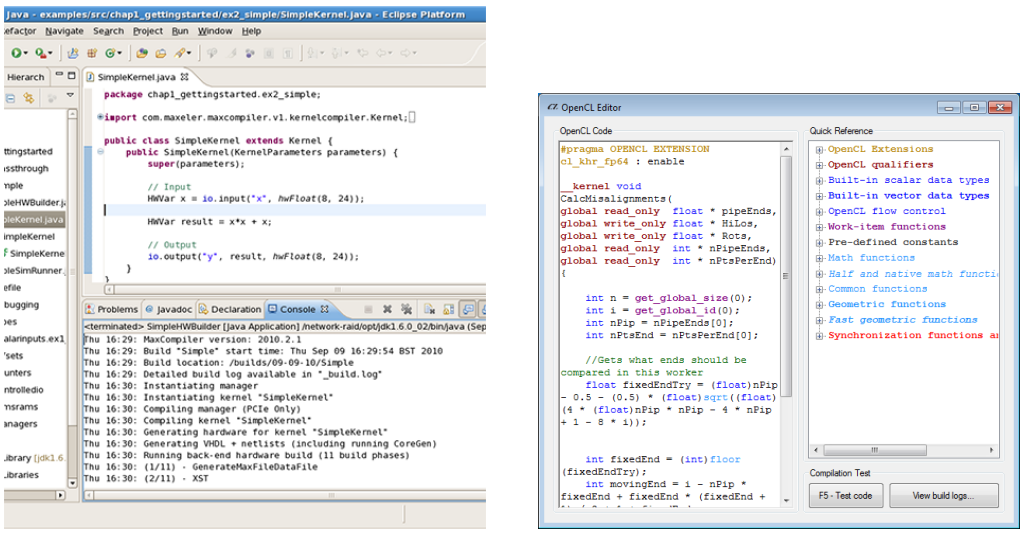


Figure 1.2: Some of many available HDLs and their infrastracture: Maxeler MaxJ with MaxIDE [3] and OpenCL [4] with OpenCL Editor [5].

operators can be implemented using custom arithmetic, offering a trade-off between accuracy and throughput. The lower the operator precision, the more coefficients can be implemented. Alternatively, lower precision operators allow for more FIR filters on a single FPGA chip. The goal of optimization is to find the design offering the highest throughput for a specified accuracy. The optimization also includes design clock frequency, making design generation noisy due to potential timing issues. Depending on the parameter configuration, the design has a smaller or larger probability of hardware generation failure. For example, the real-time Proximity Query (PQ) design [46] suffers both from Place and Route (PAR) and timing problems and as a result hardware generation often fails. Looking back at the FIR design, it involves a lot of data streaming onto the FPGA. It is possible that the computation throughput is going to be limited by the connecting bus. Despite FIR being a very simple design, the above mentioned properties make optimization challenging. An exhaustive approach could use a script such as the one presented in Figure 1.3 to explore the parameter space.

Generally optimization of designs requires the designer to analyze the design, create models and benchmarks, and subsequently use them to optimize the design for throughput, power consumption or some other performance metric. This is a time consuming process even for an experienced designer, often lasting a number of days. There are multiple other examples of similar problems, such as optimization of multi-FPGA systems [71]. The level of parallelism can have non-obvious impact on the performance of run-time

```
# clock frequency
freq = 100
# iterate over the unrolling factor
for p in [1, 2, 4, 8] :
    # iterate over width of the exponent
    for wE in range(8, 11) :
            # iterate over width of the mantissa
        for wM in range(11, 53) :
            # generate directories and hardware configuration file
            buildConfig(p, freq, wE, wM)
            # build hardware using the generated configuration
            os.system("Make hw")
            # run the generated hardware
            os.system("Make run")
            # analyze results
            results = analyze()
```

Figure 1.3: A sample script used for exhaustive search over a range of design parameters. The code generates hardware configuration, builds the design and executes a benchmark. Finally the results are analyzed.

reconfigurable designs [30]. The design throughput [46] is highly dependent on the numerical representation. It influences resource utilization and therefore the level of parallelism. Optimization of coefficients of constant multipliers can also yield improvement [76]. Balancing data-reuse and loop-level parallelism for hardware generation requires complicated frameworks [94]. Determining optimal stencil configuration is known to be a difficult problem [109]. Although authors present tools and models which allow for efficient optimization of specific designs, they have to be updated for new designs. The high manual effort and low re-usability of the tools make those approaches highly inefficient in terms of designer's productivity. Automation of tools and their generality are highly desired for increased productivity.

There were few attempts in the past to create generic automatic optimization tools for reconfigurable designs. The are two main obstacles in the engineering and application of those tools. The first obstacle is the massive design parameter spaces which possibly span out into thousands of designs. The second obstacle is the long design evaluation time, taking multiple hours to generate hardware and execute benchmarks. A prior attempt used surrogate modeling to speed-up expensive design evaluations [117]. Fitness Inheritance was used to decrease the number of design evaluations and hence speed-up optimization. In [100] authors present a design and Computer Aided Design (CAD) tool parameter tuning approach. Equally to the design parameters, proper selection of those CAD tool

```
# parameter space definition
parameters = {"freq_min" : 100,
              "freq_max" : 100,
              "p_min" : 1,
              "p_max"  : 10,
              ...
              }

# Build bitstreams and run benchmarks, the fitness function
def buildHardwareRunBenchmark():
      # execute bitstream generation
        os.system("Make hw")
      # execute benchmark and / or analyze bitstream
        return os.system("Make run")

# supply the parameter definition and scripts to the optimization algorithm
optimialDesign = Algorithm(parameters, buildHardwareRunBenchmark)
```

Figure 1.4: Automated parameter optimization algorithm. It takes as input the hardware generation scripts along with the parameter space definition.

parameters can allow for generation of better performing design. The technique offers a very high degree of parallelism, the authors generate up to 50 designs in parallel. Very few designers can build more than a few designs in parallel, let alone 10 or 50. Furthermore, the investigated case study requires only 20-25 minutes for hardware generation, an unrealistic example. Some of the proposed parameters, like random seeds used for placement, not necessarily being good candidates for optimization. Lastly, the authors claim that the presented approach offers a 20x reduction in the number of evaluated designs compared to exhaustive search. This makes the approach unrealistic for multi-parameter designs with millions of possible designs. Timing optimization using cloud computing and machine learning is presented in [80]. Again, the optimization targets CAD parameters. Multiple different designs are evaluated, and it is shown how tuning of the CAD parameters can offer better chance of design meeting timing.

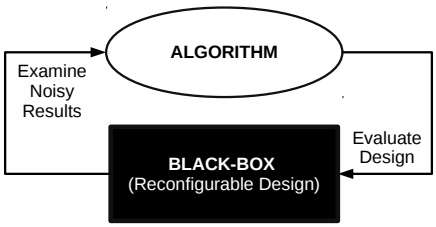


Figure 1.5: Optimization of reconfigurable designs as a black-box optimizaton problem.

Unlike previous work, our work focuses on the architecture parameters of the designs, rather then CAD tools. Our approach in principle could be applied to CAD tool parameter optimization, yet those parameters usually number in dozens and require a different approach [80, 100]. We start by treating reconfigurable design parameter optimization problem as a noisy black box global optimization problem, as shown in Figure 1.5. The noisy black box optimization problem is optimization of a noisy function with no knowledge of its inner workings. We do this to allow the designer to focus on the actual design, rather then analytical treatment of the design optimization problem. The input to the algorithms developed in this thesis is a script that for a given design parameter configuration builds it for a specific FPGA device. The script also compiles benchmarks to assess performance of the design with that configuration. Depending on the outcome, the script outputs an exit code indicating if the design with a given configuration met all of the design constraints or not. Those constrains range from resource constrains to more design specific like output accuracy. If possible, the script also outputs the performance metrics — like execution time or power consumption. A sample script is presented in Figure 1.4. This packaging of the design generation and assessment processes allows us to optimize a range of designs with different toolchains. It is suitable for any design and is very beneficial for experimental setting. For a production system, tighter coupling of the algorithms with synthesis tools and benchmarks is possible.

We show it is useful to construct surrogate models for the reconfigurable designs. As these models are orders of magnitude faster to evaluate than generation of bitstreams and code execution of benchmarks, they substantially accelerate optimization, enabling an automated generic approach. Most generic automated tools, like exhaustive search or meta-heuristics, can be quickly deployed and require little development time, yet they often lack efficiency and require hundreds to millions of fitness function evaluations. In case of reconfigurable designs this unrealistic number of hardware generations, making them impractical. We show that this is not necessarily the case. Our generic tools offer similar optimization time to manual optimization, often finding the globally optimal design configuration. As an example, the script to optimize the FIR design, as presented in Figure 1.3, would be rewritten as shown in Figure 1.4. This is the approach we follow in Chapter 3 and 4.

Furthermore, we show how the tools can learn from one optimization onto another. Currently optimization attempts only contribute to the designer's experience, they are not recorded and do not contribute towards more efficient future development by other designers or tools. We investigate the potential of storage and transfer of knowledge

```
1
2  # design optimization repository
3  design_db = {[100, 1, 11, 53] : [55.0], # Previously optimized related design
4               [100, 4, 8,  53] : [22.0], # and data gathered during its
5               ...                        # optimization is supplied to the
6              }                           # algorithm.
7
8  # parameter space definition
9  parameters = {"freq_min" : 100,
10               ...
11               }
12
13 # Build bitstreams and run benchmarks, the fitness function
14 def buildHardwareRunBenchmark():
15       # execute bitstream generation
16         os.system("Make_hw")
17       # execute benchmark and / or analyze bitstream
18         return os.system("Make_run")
19
20 # supply the parameter definition and scripts to the optimization algorithm
21 optimalDesign = Algorithm(parameters, buildHardwareRunBenchmark, design_db)
```

Figure 1.6: Automated parameter optimization extended with a design optimization repository. Extra information can improve both the speed and accuracy of the optimization.

gained during design synthesis for acceleration of future design optimization. In particular, we show how results of implementation of a design onto a platform can automatically be used for faster optimization of the same design implemented on a different device. We also show how results of optimization of a design can aid optimization of another design with similar architecture. We particularly investigate knowledge transfer in the context of surrogate model based optimization, as presented in Figure 1.6, in Chapter 5.

1.2 Contributions

The main aim of this thesis is to *provide tools and techniques to allow designers to automatically optimize reconfigurable designs, particularly FPGA based, while minimizing required experience and input from the designer*. By optimization we refer to the problem of reconfigurable design parameter optimization with respect to benchmarks measuring design throughput, latency, energy efficiency or some other metric. The problem is treated as a noisy global black-box optimization problem. This prevents us from making any strong assumptions about the design and allows for a generic approach. The reconfigurable designs studied are based on systems consisting of a CPU(s) and a FPGA(s). There

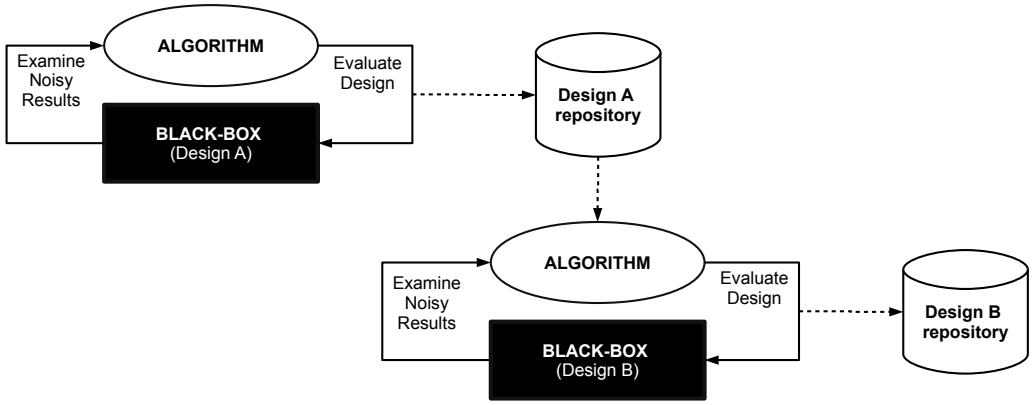


Figure 1.7: Knowledge transfer in black-box optimization of reconfigurable designs.

are three main contributions in this thesis. This thesis has three main contributions to address the challenges shown in Table 1.1

1. Traditionally, optimization involves manual design analysis, modeling, and exploration tool creation. This requires an experienced designer, and is a time consuming process. The optimization time is challenging due to long hardware generation time. We develop the Machine Learning Optimizer (MLO) algorithm to automate this process. From a number of benchmark executions, we automatically derive the characteristics of the parameter space and create a surrogate model of a fitness function through regression and classification. Based on this surrogate model, design parameters are optimized using metaheuristics. The new algorithm can improve optimization time by up to 50% compared to design specific optimization tools. Compared to standard hill climbing algorithm it offers between 3 to 5 times faster optimization time while discovering better performing design configurations. The work is published [89, 91].

2. The previous approach requires tuning and is sensitive to its configuration. Based on the previous findings, we develop Automatic Reconfigurable Design Efficient Global Optimization (ARDEGO) algorithm to improve over MLO its performance and simplify the optimization loop. This reduces further the required user input and experience. The new algorithm is suitable for problems with large number of parameters. Furthermore, to speed-up optimization time the algorithm is parallelized. The algorithm is shown to optimize a 7-dimensional stencil-based design in less than 93 hours. When using multiple worker nodes during optimization of a financial quadrature-based design a 22% reduction in optimization time is observed compared

to the design specific optimization tool. The work is published in papers [47, 88].

3. Both of the previous approaches do not learn from previous optimization. There is a strong indication that reusing previous optimization results can substantially speed-up new optimization attempts. We show how the knowledge gathered during optimization attempts of similar designs, or designs ported across platforms, can be transferred to speed-up optimization and improve the final design performance. By using knowledge transfer optimization time is reduced by up to 35% compared to ARDEGO.

Table 1.1: The three major challenges with optimization of reconfigurable designs.

Challenge	Description
Hardware Generation Time	Generation and evaluation of a single parameter setting takes between an hour to two days.
No. of possible configurations	The curse of dimensionality. The number of potential design configurations can be in the millions.
Lost Knowledge	The knowledge gathered during development and optimization of designs is not transferred.

1.3 Overview of the Thesis

The thesis is organized into six chapters. This chapter consists of an introduction and motivation behind the work presented in the thesis. Chapter 2 provides the reader with the background information necessary to understand further chapters. Each of the Chapters 3, 4 and 5 deal with one of the previously mentioned research challenges. Chapter 6 presents future work and conclusions.

The main focus of the thesis is to reduce the effort associated with reconfigurable design development by automating parameter optimization. The designer should focus on the design development and correctness, the tools should carry out the optimization. In **Chapter 3** we present the initial investigation of reconfigurable design optimization using the MLO algorithm. The algorithm is based on the surrogate model technology and particle swarm optimization. The method shows a lot of promise, yet it suffers from a number of problems; complicated optimization loop and lack of parallelism. The complexity of

optimization loop is investigated by examining the impact of MLO configuration on design optimization performance. MLO also struggles to optimize designs with larger number of parameters.

In **Chapter 4** we focus on decreasing the effort associated with optimization of reconfigurable design. Based on the experience gathered during MLO evaluation, a new algorithm is presented, called ARDEGO. The new algorithm is based on the Efficient Global Optimization (EGO) algorithm and it offers several improvements over MLO. It offers asynchrnous parallelism, and a simpler optimization loop. Hardware generation time is random, and two designs can take different amount of time to generate. To ensure efficiency asynchrnous parallelization is necessary. As the ARDEGO algorithm uses multiple optimization nodes, its optimization speed is greatly increased relative to MLO.

The main drawback of ARDEGO compared to a human designer is that the algorithm does not build up its knowledge with subsequent design optimization. **Chapter 5** presents Auto-Transfer algorithm and shows how results of design synthesis and benchmarking can be transferred to decrease optimization of a similar design when ported onto a different platform or when its code is revised. In modern dynamic environment the designs are often modified and the underlying platforms revised. A methodology to transfer old synthesis results is crucial to automation.

In **Chapter 6** we present future work and conclusions. Possible extensions and research direction are presented.

Chapter 2

Background

This chapter presents existing concepts and approaches required to understand the work presented in this thesis. Most importantly, we formalize the optimization problem of reconfigurable designs.

Initially, we present the concepts behind reconfigurable designs and their development. Then we overview some mathematical optimization techniques and provide formal definitions of the reconfigurable design optimization problem. This is followed by an introduction to the machine learning techniques used in this thesis. Lastly, combining all the previous concepts, surrogate modeling is introduced.

2.1 Reconfigurable Designs

An FPGA is a semiconductor device that allows the programmer to emulate a digital circuit or a program provided that it fits within the chip resource limit [10, 82, 138]. The device was invented by Xilinx Inc. in 1980's [12, 159]. FPGA can be used as a standalone computing device or as a sub-component of a heterogeneous systems. Throughout most of history, it was Xilinx Inc. and Altera Corporation that were the two main FPGA vendors [13]. The growing importance of heterogeneous FPGA systems being signaled by recent acquisition of Altera by the Intel Corporation [14]. There is even a strong indication for potential for future hybrid ASIC-FPGA architectures. Furthermore, in recent years the number of FPGA design start-ups have been drastically increasing. This is mainly due to the advantages FPGA offer over ASIC based technologies, listed in table Table 2.1.

FPGA designs are far more suitable for low scale systems, where the cost of developing an ASIC chip would outweigh the potential gains. The cost of producing a single FPGA chip is much higher, but designing an efficient ASIC chips takes months, if not years,

Table 2.1: FPGA and ASIC comparison [10, 49, 82, 87, 138]

ASIC	Reconfigurable Hardware
1.Full customization	1.Short time to market
2.Lower unit costs	2.Low upfront cost
3.Smaller form factor	3.Simpler design cycle
4.Higher clock cycle	4.More predictable project cycle
5.Very high upfront cost	5.Programmability

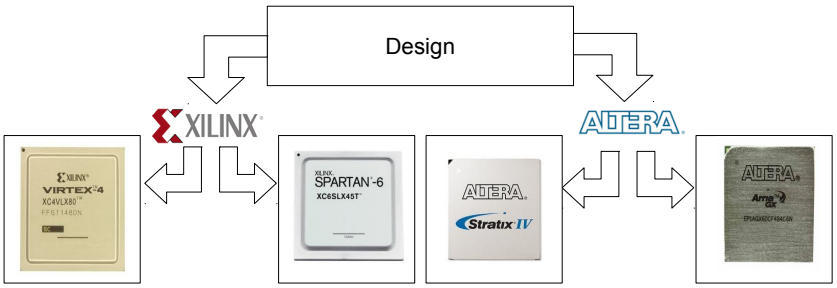


Figure 2.1: FPGA design and a small subset of available platforms. Courtesy of Altera Corporation [6] and Xilinx, Inc. [7].

whereas designing an FPGA design can take days (given that one is only customizing a ready design). Even a full scale FPGA design is less time consuming than ASIC, mainly due to lower prototyping cost. Furthermore FPGA can be upgraded relatively cost free, while ASIC requires the procurement of new hardware or complicated software patches. These characteristics made FPGA an interesting remedy to many problems; often offering superior performance and task customization at a much lower cost.

In principle, any design built for ASIC can be implemented on any FPGA. The main challenge is to make effective use of limited FPGA embedded resources. Most commonly a FPGA device is composed of Lookup Tables (LUTs), Digital Signal Processors (DSPs), flip flops, Block RAMs (BRAMs), and Debug Support Units (DSUs) [55, 82]. To make matters more worse, vendors often offer FPGAs with more specialized embedded resources. The resulting FPGA architecture is very complicated, hence why a layer of hardware abstraction is essential. As presented in Figure 2.1 a design can be mapped onto different FPGA platforms offered by various vendors. The design architecture should focus on

utilizing generic FPGA features, and the mapping should focus on implementing the components efficiently. Similar in spirit to software where the compiler generates computer specific machine code.

Although FPGA designs can be ported across different devices, they vary in the degree of portability. For example the Xilinx MicroBlaze soft processor [15] makes heavy use of vendor hardware libraries and as a result only targets specific reconfigurable devices. Vendor and chip specific hardware libraries provide a large number of circuits like memory controllers, Ethernet controllers or others [16, 17]. Fully cross-vendor and cross-platform portable design cannot be based on hardware libraries and has to be separated from the underlying device, a good example is high-frequency trading FPGA library [98]. The use of hardware libraries introduces a trade-off between portability and performance and is an important aspect of FPGA design. To summarize, FPGA designs have two main characteristics:

Customizability - FPGA designs can be customized and become configurable with parameters, in the same manner as objects are instantiated with different variables. Some possible parameters are numerical representation [46, 155], constants coefficients [76], using different level of parallelism [30] or stencil configuration [109]. The fact that FPGAs are clocked in the range of MHz instead of GHz and consist of far smaller number of logical gates than ASIC designs means that in a large number of cases they are going to be orders of magnitude slower than their counterparts. That is an obvious disadvantage, which is countered by customization to better fit the goal performance levels, lower cost and time to market.

Portability - FPGA designs are often portable, although some physical constraints can limit that. Different LUT or interconnect architecture can allow for efficient implementation on one FPGA device while on other it might waste resources as its sub-components were designed to exploit hardware based multipliers. The vast majority of FPGA designs are developed using Intellectual Property (IP) cores what allows for efficient cross platform migration of a design, given the new platform supports required IP cores [98]. This is analogous to writing cross platform software. Others are designed through HDL generation scripts, which can generate code for different platforms [165]. Some applications utilize architecture-specific properties making them inefficient out of their target systems range - while others are designed to be used across a number of different platforms. The concept goes as far as creating virtualization of FPGA platforms [83].

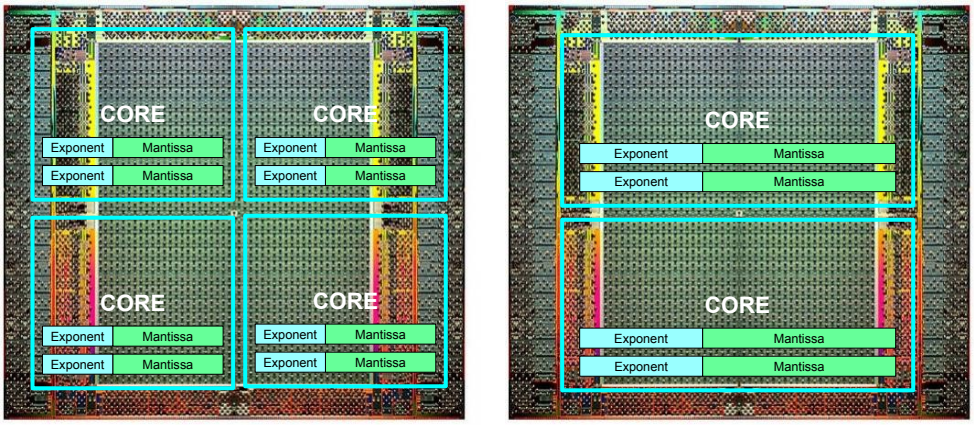


Figure 2.2: Two configuration of a reconfigurable design. One configured with 4 cores and narrower numerical implementation, one with two cores and wider numerical representation.

2.1.1 Applications of FPGAs

FPGAs are employed as either an ASIC prototype, or as a target computation device. Although ASIC designs offer better performance, the customizability, portability and lower development cost make FPGAs attractive computation devices.

Large ASIC developers, like Intel or AMD, often employ FPGA arrays to emulate their new CPU or other devices. This is called ASIC prototyping. Simulation of complex digital circuits is time consuming, especially if one requires great level of detail. FPGA offers the advantage of being able to interact with real hardware components emulating ASIC chips with little computational overhead.

FPGAs can be used for computation either as stand-alone devices or as part of a heterogeneous system. In stand-alone mode all of the computation is carried by the FPGA. FPGAs are used as stand-alone devices like drones [59] or reconfigurable radios [30], where they can be optimized for power saving. In a software defined-radio system a large chunk of computation is decoding and encoding of audio signals. FPGA allows for high throughput system that can be configured to work with any new or legacy standard that is needed. In a heterogeneous system a portion of an application code is implemented as a digital circuit on the FPGA, while the rest is run on a CPU or some other device [23, 27, 143]. Heterogeneous systems are often used by the High Performance Computing (HPC) community to carry out intensive computation. Depending on the subtask, either CPU or FPGA carries computation. Quite often, the CPU is used more

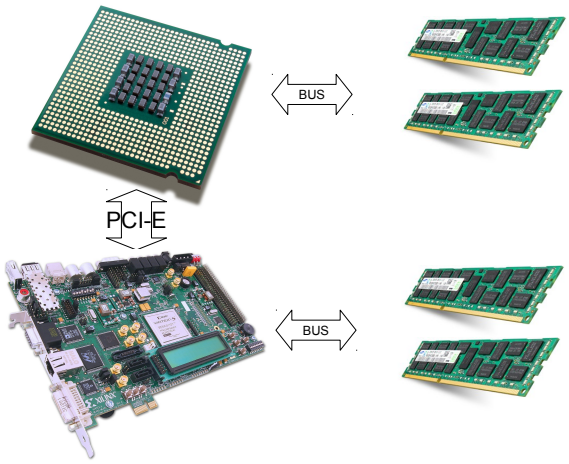


Figure 2.3: FPGA Heterogeneous System. Courtesy of Intel Corporation [8], Samsung Electronics Co., Ltd [9] and Xilinx, Inc. [7].

for the administrative tasks, while the FPGA is the main computation engine. There are numerous examples of applications where FPGA are used to achieve better performance. They have been extensively used in finance, solving problems like low latency data feed handling [119] to option pricing and other problems [43, 77, 153, 155]. Geophysics [109] and bioinformatics [92, 142] are another areas where FPGAs have been extensively used.

2.1.2 Reconfigurable designs as computation devices

FPGAs are commonly used as computation devices where portions or the whole application is used for computation. Typically a board containing one or more FPGA chips and associated DDR memory is installed in a PCI-E slot. The system can be heterogeneous if both the CPU and FPGA are involved in computation. Vendors usually provide an Application Programming Interface (API) and a tool-chain to develop and use bitstreams with HLL. These can vary substantially in quality and complexity, depending on user needs and their budget, nevertheless the design schematic is similar. An application can be designed from scratch using FPGA, or if already available accelerated, the principles and development approach remains unchanged.

Acceleration over homogeneous computing system is usually achieved by porting the computationally intensive loops onto the FPGA board/s utilizing streaming computation paradigms. In streaming model, the calculations are done by performing operations on continuous data streams instead of executing instructions. Figure 2.4 presents a for loop

with a trivial streaming mode implementation. On an instruction based computer where the control overhead is significant in terms of silicon and time, one has to perform a number of memory access, caching and many other operations requiring between 10 and 100 clock cycles to get a single z value calculated. On a custom machine one can create a design that will increment and add the two data streams producing a result every clock cycle using small amount of silicon. Designs utilizing this approach can achieve substantial speed-up as memory latency is hidden and application control is severely limited per useful bit of computation.

```
1
2 for i in range(0, N):
3     z[i] = x[i] + y[i] + 1.0
```

Figure 2.4: Simple loop suitable for FPGA based heterogeneous system.

2.2 Design Development

A design is an implementation of a program using a reconfigurable platform. It consists of FPGA suitable HDL description and software. The software component is typically used to transfer the data from and onto a reconfigurable platform, as well as to initialize the computation and control some behavior of the reconfigurable platform. Figure 2.4 represents a simple design, it performs addition of two time series.

The development usually begins by coding the design and benchmarks used to assess its quality. It is also possible for the designer to use a High Level Synthesis (HLS) language to describe the design [49]. HLS involves HDL description of an application which is synthesized into a hardware design. The idea is analogous to the idea of a software compiler. They are important in the context of reconfigurable computing as they largely make the underlying platform transparent to the programmers. Through the use of libraries and integrated tool-chains, hardware compilers provide an programming model API. Good examples are ROCCC 2.0 [18], YAHDL [39], OpenCL [4] and MaxCompiler [3]. Instead of HDL they use an exotic flavor of Java (MaxCompiler), C (ROCC) or some other HLL to describe the FPGA design. Another example is a semi-automated Python code hardware acceleration where the user selects which loops are to be ported onto FPGA [135]. In [49] a comparison is made between AutoESL's AutoPilot HLS tool [166] based approach and manually optimized designs. Those automated approaches are analogous

to the CUDA HLL interface provided by Nvida to access the computational resources of their Graphic Processing Unit (GPU) processors [108]. A sample two loop design is presented in Figure 2.4. The loop is not as easily accelerated as the one presented in Figure 2.5. The second loop depends on calculation of *mean*, which can only happen after the loop has finished executing. Usually the data flow dependencies are far more complex, making the development task complicated.

```
1  acc = 0.0
2  for i in range(0, N):
3          for j in range(0, M):
4                  acc += matrixA[i][j]
5
6  mean = acc / (N*M)
7
8  for i in range(0, N):
9          for j in range(0, M):
10                 matrixB[i][j] = matrixB[i][j] - mean
```

Figure 2.5: Two loop code.

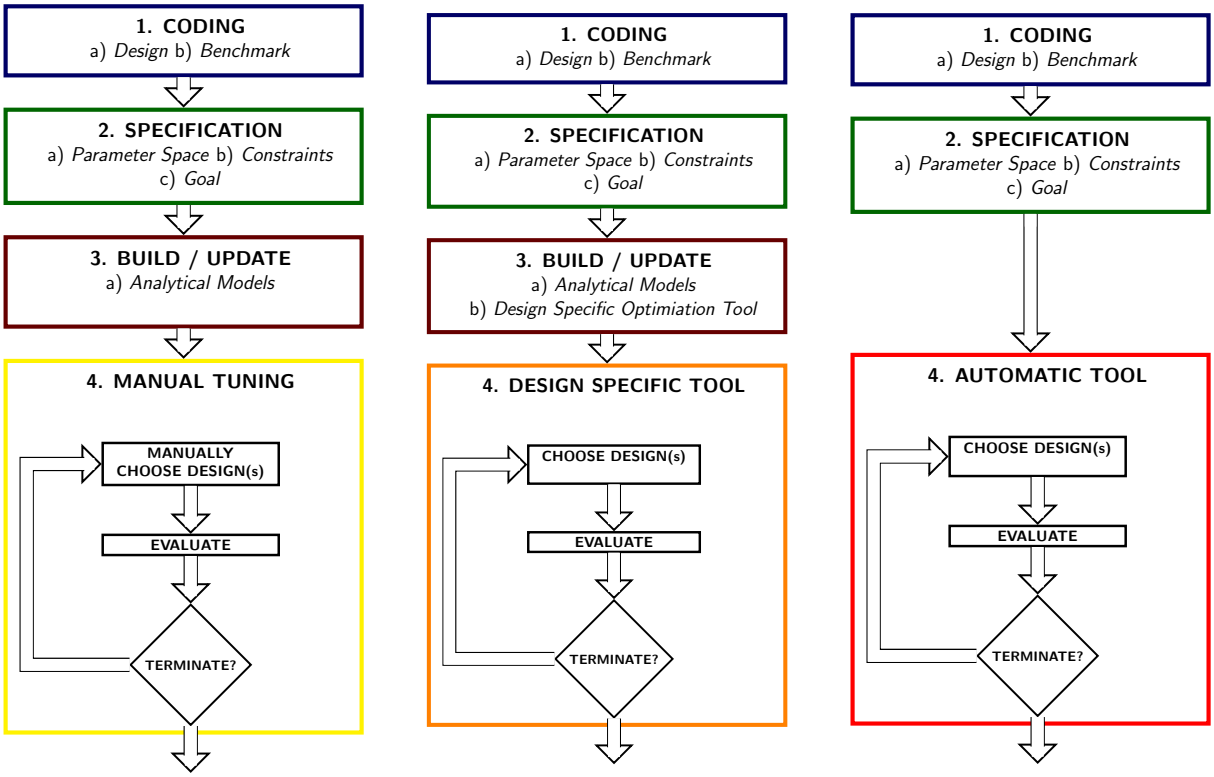
2.2.1 Design Development and Optimization Approaches

Optimization begins once the designer managed to create a basic functional version of the design. Traditionally, optimization of reconfigurable designs is carried out by building benchmarks, deriving analytical models, and creating optimization tools [155, 30, 76, 94]. This allows for a manual, design specific automatic optimization approach, or a generic optimization as presented in Figure 2.6.

Typically manual optimization approach relies on the experience of a designer. The designer builds the design, and uses his experience and intuition to optimize the design. This is often done by modeling the performance of the design, and building a limited number of bitstreams. It is the preferred method when parameter space spans large number of designs or is difficult to model. The designer can navigate in the design space examining hardware generation and benchmark output. It does require an experienced designer and is labour intensive. Examples of such optimization are [109, 46].

The design specific automated approach is a step forward in automation from the manual approach. The designer create analytical models to predict design performance as well as design specific optimization tools. The workload overhead can be substantial, however, it is not as labor intensive as the manual approach as the parameter space exploration is automated. It allows for the optimization tool to re-optimize designs when

constraints change. It also relies on the designers experience. Examples of design specific approaches are [155, 30, 76, 94].



(a) *Manual* (b) *Design Specific Automatic* (c) *Generic Automatic*

Figure 2.6: Different optimization approaches.

In a generic algorithm the parameter exploration is performed by an automated tool which uses user-provided scripts. An example of a generic optimization approach is an exhaustive search, with the evaluation of all possible designs. To use exhaustive search, the designer prepares a script which builds and benchmarks a design. Exhaustive search is a simple loop over the parameter space. It takes a parameter configuration, and executes the script. Then it collects the results. It continues until the whole space has been traversed. It has the advantage that it can be applied to any design optimization problem. Unfortunately for design spaces spanning hundreds of designs, an exhaustive search becomes too time consuming and is rarely employed. In a generic algorithm, exhaustive search can be replaced with a more sophisticated logic, like surrogate model based algorithms. The surrogate model based approach does not require the designer to analyse the application and construct analytical models. The surrogate model is integrated with the algorithm allowing for full optimization automation. Aside of coding the design, it does not rely on designer experience to perform optimization. There are numerous

examples of such approaches.

One of the earliest mentions of a generic optimization algorithm for reconfigurable designs is based on a genetic algorithm [117]. The expensive fitness function evaluations are replaced with a regression model and the cost of evaluation is mitigated by a fitness inheritance scheme, which allows to reduce fitness function evaluations. The fitness function is based on analytical models. The goal of optimization is to find a design optimal in terms of area and latency. The algorithm does not deal with problems like PAR.

A more recent generic optimization algorithm is used for CAD tool and design parameter optimization [100]. The algorithm is based on a regression model of design's performance function. In an iterative fashion it samples configurations predicted to perform best according to the regression model. One of the drawbacks of the algorithm is that it potentially can suffer from over exploitation, only evaluating designs predicted to have best performance. Another potential problem with the approach is only a 20x reduction relative to the number of potential configurations that have to be evaluated. This implies evaluation of millions of configurations for larger parameter spaces. The advantage is that its optimization loop is simple and offers parallelism.

Cloud computing combined with machine learning also seems to offer a lot of promise for generic optimization algorithms [80]. The optimization goal is to use optimal CAD tool parameter configuration for the reconfigurable design to reach timing closure. The difficulty is long evaluation time of reconfigurable designs and large CAD tool parameter space. The algorithm is designed to harness parallelism in the form of cloud computing. It uses a naive Bayes classifier to learn about the most influential CAD tool parameters. There is a potential problem in the approach stemming from the assumption that CAD tool parameters are conditionally independent of each other, which is not necessarily true. Combinations of CAD tool parameters settings can significantly affect the end result. An interesting idea of parameter space reduction through offline learning is also presented. The set of CAD tool parameters is fixed for a given device, hence a database of all optimization runs can be constructed. This database is then analysed using Principal Component Analysis (PCA) [35] picking the CAD tool parameters which have the highest impact on timing of a design. The authors mention potential problems of construction of such a database due to the high cost of configuration evaluation.

In all three of the earlier mentioned optimization approaches, manual, design specific and generic, the designer has to code the design, specify the parameter space, constraints and goals. The resulting design code and design specification is used in all three of the cases. The great advantage of an automated algorithm is that it simplifies the optimization

flow. The designer saves time on coding of the optimization tools and building analytical models, the algorithm is used instead. Guaranteed that the algorithm offers comparable optimization time, this allows for improved designers productivity. Assuming coding of the design and benchmark (step 1) is unavoidable and specification of the parameter space and optimization goal (step 2) is often straight-forward, the main issues in optimization are tasks outlined in steps 3 and 4. Creation of analytical models and tools is labor intensive, while using optimization tools is time consuming. Success of step 4 is highly dependent on step 3. In an automated approach, the user supplies a benchmark along with constraints and goals, and the the algorithm automatically carries out the optimization.

2.2.2 Reconfigurable Design Optimization Problem Statement

The optimization of reconfigurable designs is a time-consuming process, and automation is highly desired. The designer starts by describing the design and coding of benchmarks. Benchmarks evaluate reconfigurable design's parameter configurations, which is a time consuming process and often involves hardware generation and software execution. The output of a benchmark is a noisy performance measure; execution time, energy or any other target quality. This measure is called fitness in the optimization literature. In the case of reconfigurable designs the fitness function f represents the behavior of the benchmark, and vector $\mathbf{x}$ is the parameter configuration within the parameter space $\mathcal{X}$ with D dimensions (parameters).

Once the benchmarks and the design are ready, the designer defines the parameter space and the design constraints. The space defines the architecture and the physical settings of the FPGA design. If the design fails any of the constraints, the benchmark informs the designer about the error using an appropriate exit code t [89]. Many possible constraints exist; for example, the design has to fit specific area, timing and power constraints. A good example of a target design is the previously presented FIR filter performing the following function

$$f(\mathbf{x}) = \sum_{i=0}^{q} \mathbb{b}_i \times \mathbf{x}_{q-i} \tag{2.1}$$

with a batch of q filtered elements $\mathbf{x} = \{x_i\}_{i=1}^{q}$, where $x_i \in \mathbb{R}$, being processed using q coefficients $\mathbb{b}_i \in \mathbb{R}$ at every time step. Some of the possible parameters is the mantissa and exponent width of the floating point operators, unrolling factor, number of cores and clock frequency. Numerical precision and resource limitations are good examples of

constraint functions. The optimization goal could be to find the most energy efficient or highest throughput configuration.

Analytical models are constructed to predict the performance of a design. Development of these performance models often requires high levels of expertise [71, 109]. Usually an optimization tool is developed to explore the parameters space using supplied benchmarks. The optimal parameter setting is determined based on the space specification and constraints using the models to minimize the optimization time. The optimization problem of reconfigurable designs can be formally defined. Beginning with the observations of performance of a reconfigurable design y, defined as follows

$$y = f(\mathbf{x}) + \epsilon, \quad \epsilon \sim \mathcal{N}(0, {\sigma_n}^2). \tag{2.2}$$

with discrete parameters $\mathbf{x}$ belonging to the parameter space $\mathcal{X}$. The parameter space is a discrete metric space with D parameters $\mathcal{X} \subseteq \mathbb{R}^D$. The y value is the noisy observation of performance of a design obtained through evaluation of a benchmark, the process used to assess the performance is called the fitness function $f : \mathcal{X} \to \mathbb{R}$. For simplicity, additive Gaussian noise with mean 0 and variance σ_n^2 is assumed. Along with design performance, a benchmark evaluates constraints using a non-deterministic function c and returns performance $f(\mathbf{x})$ and a discrete number $c(\mathbf{x}) = t$, indicating which constraints are not satisfied. The observed exit code t indicates constraint violation if $t \neq 1$. The set of possible exit codes is denoted as $\mathcal{T}$. For example, for a particular design $t = 1$ would indicate no constraint violation, $t = 2$ would indicate inaccurate design, $t = 3$ failed timing constraints and $t = 4$ a dual, accuracy and latency, constraint failure. A constraint is either one of k equalities g or one of r inequalities h:

$$g_i(\mathbf{x}) = \mathfrak{c}_i \quad i = 1, .., k. \qquad\qquad h_j(\mathbf{x}) \leq \mathfrak{c}_j \quad j = 1, .., r. \tag{2.3}$$

where $\mathfrak{c}_i$ and $\mathfrak{c}_j$ are some constants. The constraint violations are encoded in a c function to indicate which, if any, have failed for a given $\mathbf{x}$. The constraints can depend on a random process and are not guaranteed to be deterministic. In general, those random process are unknown.

For simplicity, it is assumed that c is a random process, that consists of a number of nonidentical independent random variables, each following a distribution with an unknown probability density function $\mathfrak{p}_{\mathbf{x}}(t) = \mathrm{p}(c(\mathbf{x}) = t)$. With probability one the outcome is one

of the possible exit codes

$$\forall \mathbf{x} \in \mathcal{X}, \sum_{\forall t \in \mathcal{T}} \mathfrak{p}_{\mathbf{x}}(t) = 1. \tag{2.4}$$

For two neighboring configurations $\mathbf{x}$ and $\mathbf{x}'$, with respect to the Euclidean distance $|\mathbf{x} - \mathbf{x}'|$, the Kullback-Leibler divergence [86] $D_{\mathrm{KL}}(\mathfrak{p}_{\mathbf{x}}||\mathfrak{p}_{\mathbf{x}'})$ and $D_{\mathrm{KL}}(\mathfrak{p}_{\mathbf{x}'}||\mathfrak{p}_{\mathbf{x}})$ is usually small. The Kullback-Leibler divergence is intuitively understood as a measure of the information lost when using one distribution to approximate another. In this case, it is as a statement, saying that the distributions are similar. The reasoning is that nearby configurations exhibit similar probability of constraint satisfaction. For example, a 4 core design uses 53 bits for numerical representation and has a 95% chance of satisfying all constraints. The probability is most likely going to be $\approx 95\%$ for the same design configured with 4 cores and a 52 bit numerical representation.

The fitness and classification functions are encapsulated in a benchmark function b.

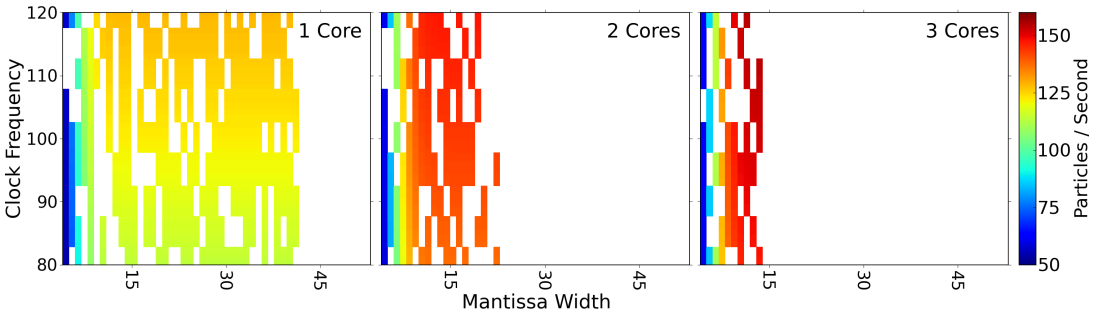
$$b(\mathbf{x}) = (f(\mathbf{x}) + \epsilon,\ c(\mathbf{x})). \tag{2.5}$$

The optimization objective is to find an optimal parameter setting $\mathbf{x}_{opt} \in \mathcal{X}$, which optimizes the fitness $f(\mathbf{x})$ while not violating any constraints through multiple evaluations of benchmark b. The set of admissible parameters, which does not violate any constraints and for which f is defined is called valid region $\mathcal{V}$. The set $\mathcal{V}$ is defined as the subset of the parameter space that has a non-zero probability of satisfying all constraints

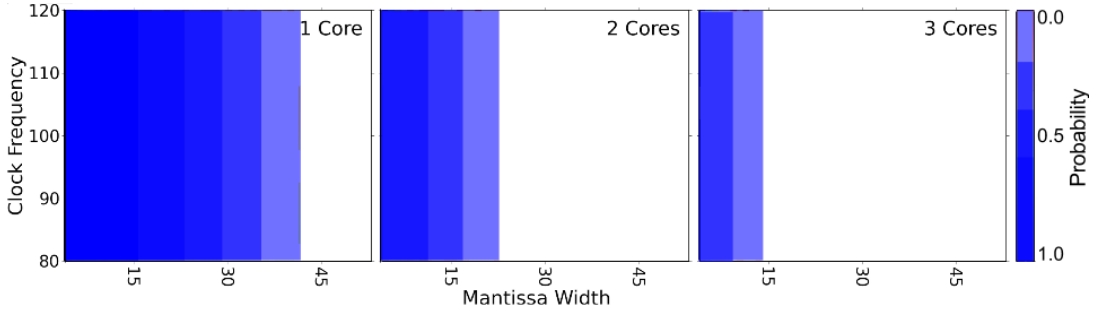
$$\mathcal{V} = \{\mathbf{x} | \exists \mathbf{x} : [\mathfrak{p}_{\mathbf{x}}(1) > 0] \wedge [\mathbf{x} \in \mathcal{X}]\}. \tag{2.6}$$

For example a configuration, which has a 50% probability of generating hardware that meets timing resides in the valid region. A configuration which violates a deterministic accuracy constraint does not belong to the valid region. Those configurations, which have 0% probability of meeting constraints, belong to the invalid region $\mathcal{I}$ of the parameter space. A good example is a configuration which overmaps on resources.

$$\mathcal{I} = \{\mathbf{x} | \exists \mathbf{x} : [\mathfrak{p}_{\mathbf{x}}(1) = 0] \wedge [\mathbf{x} \in \mathcal{X}]\}. \tag{2.7}$$



(a) *Fitness function*



(b) *Probability of successful hardware generation*

Figure 2.7: PQ design throughput fitness function [46] (a) and probability of succesful hardware generation visualization (b). Image (a) is based on real hardware generation, the design was implemented for Maxeler MPC-X1000 system with a Xilinx Virtex-6 XC6VSX475T FPGA. White patches represent unsuccessful hardware generation. In (b) green line marks the boundary between the valid and invalid region, white area is the invalid region. The probability figure is for demonstration purpose only, the numbers do not demonstrate the actual chance of successful hardware generation.

The global optimization problem of reconfigurable designs is then defined as finding the best possible configuration

$$\mathbf{x}_{opt} = \underset{\mathbf{x}\in\mathcal{V}}{\operatorname{argmax}}\, f(\mathbf{x}). \tag{2.8}$$

Taking into account the previously defined noisy nature of f, it is a global optimization of a noisy black box function. A sample problem is presented in the Figure 2.7. It presents the PQ design [46] with PAR, timing and resource issues. The valid region shrinks due to resource constraints as the number of cores is increased. Due to the random nature of the PAR process, random discontinues are visible in the fitness function visualization. Similar formulation of optimization problems with random constraint functions is presented in [64]. The main difference lies in the formulation of the constraint function, where the authors assume only inequality constraint functions and do not explicitly provide information on

the behavior of the constraint function.

The formulation of the problem of reconfigurable design parameter optimization forms the basis of this thesis, it is addressed by Chapter 3, 4 and 5. Extensions to the problem are presented in Chapters 4 and 5. The formalization should allow the reader to better relate the problem to the Bayesian optimization literature; justifying current treatment of the problem and spanning future research.

2.2.3 Parameter Space

The parameter space $\mathcal{X}$ of a reconfigurable design is a metric space determining both the architecture and physical settings of FPGA designs. The Euclidean distance $|\mathbf{x} - \mathbf{x}'|$ is a meaningful metric for the parameter space. The below definitions describe commonly understood definitions of reconfigurable design's parameters.

Categorical A parameter consisting of a set of values with no meaningful metric. A set consisting of FPGA chip A and FPGA B would be an example.

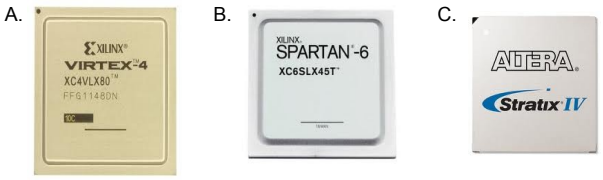


Figure 2.8: Categorical parameter. Courtesy of Altera Corporation [6] and Xilinx, Inc. [7].

Continuous A possibly bounded subset of $\mathbb{R}$. An example of a continuous parameter is clock frequency.

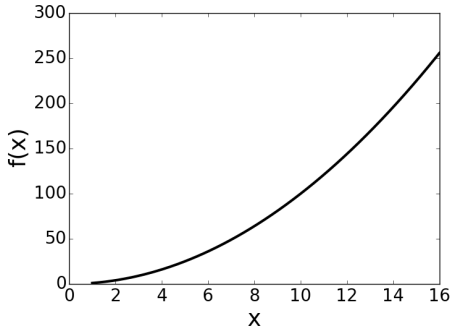


Figure 2.9: Continuous parameter.

Uniformly Discrete A possibly bounded subset of $\mathbb{R}$ consisting of isolated points. The uniformly discrete parameter consists of a subset of $\mathbb{R}$ where all points x_i's are equidistantly spaced. An example is number of computational pipelines.

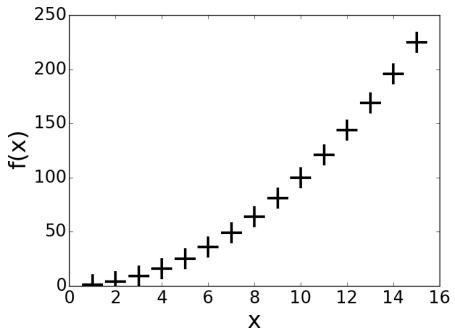


Figure 2.10: Uniformly discrete parameter.

Non-Uniformly Discrete A possibly bounded subset of $\mathbb{R}$ consisting of isolated points which are not equidistant. Degree of parallelism or number of cores that is scaled up by powers of two are good examples. Another example is a parameter space consisting of a set of divisors.

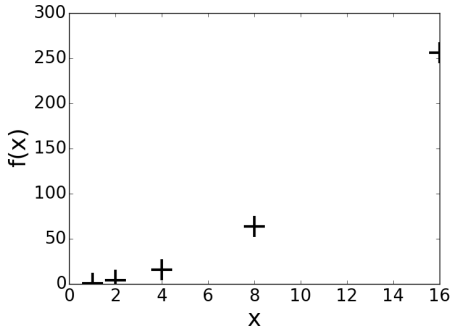


Figure 2.11: Non-uniformly discrete parameter.

The majority of the parameters of reconfigurable design parameters are uniformly discrete, even if they appear continuous. The designer chooses a step size s and transforms the continuous space into a uniformly discrete space. The step size s should follow the physical properties of the parameter. For example, if clock frequency parameter can be adjusted in 500 KHz steps, that is the stepsize. If mantissa width can be adjusted in bit increments, one bit is a step size. The categorical parameters are generally not part of the parameter space used in optimization. They are represented by sets over which Euclidean distance is not be defined. Different types of FPGA (Virtex 6, Stratix V, etc..) or different

implementations of memory controllers (controller A DDR 400 Mhz, controller B DDR2 333 Mhz, controller B DDR2 400 Mhz, etc..) are examples of unstructured parameters.

Table 2.2: Summary of the most commonly encountered parameters.

Parameter	Examples
Continuous	Clock Frequency (not strictly continuous), software parameters,..
Uniformly Discrete	Number of cores, Number of pipelines, Mantissa width, Exponent width, Number of integer bits, Unrolling factor,..
Non-Uniformly Discrete	Number of cores, Number of pipelines
Categorical	Memory controller, FPGA chip, etc..

2.2.4 Fitness Function and Constraints

Given a parameter setting $\mathbf{x}$, the benchmark $b(\mathbf{x})$ returns two values: y, the observed design fitness and t, the exit code indicating if design met all constraints. Execution time and power consumption are examples of fitness measures. There are many possible exit codes t, with 0 indicating valid parameter settings $\mathbf{x}$. 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.

Table 2.3: Examples of fitness function constraints.

Constraints	Explanation
Timing	Depending on the design, the chip and clock frequency it might be difficult to synthesize a design to reach the required timing.
PAR	The PAR process depends on the design and the chip. Generally, the higher the resource utilization the more difficult it gets to PAR a design.
Resource	FPGA chip has limited resources. Bigger designs often struggle to fit onto a chip.
Accuracy	Depending on the requirements and numerical representation used in the design, the design might produce inaccurate results. This is usually adjusted by increasing the number of bits allocated to numerical operators, increasing accuracy at the expense of resource and complexity.

There are two types of exit codes. The first type indicates a valid parameter setting. The second type indicates a configuration that possibly resulted in hardware being generated yet failed at least one constraint. The region of $\mathcal{X}$ that defines parameters settings $\mathbf{x}$ that produce $f(\mathbf{x})$ and satisfy all constraints is defined as valid region $\mathcal{V}$, region with the designs failing at least one constraint is called the invalid region $\mathcal{I}$. It is important to take into consideration random nature of the c function. The probability of design meeting timing score or being able to PAR is largely random, although correlated with the parameter setting $\mathbf{x}$. The higher the clock frequency, and the higher the resource utilization, the harder it is to generate the design. The probability of design generation is further influenced by the number of cost tables and design effort set to generate hardware.

Function f does not have to be bounded, is very or completely non-smooth, continuous or discontinuous and noisy. There are numerous examples of exponential, quadratic, linear and other behavior of fitness functions across dimensions. The discontinuities of f over $\mathcal{F}$ arise from bottlenecks, and over $\mathcal{X}$ from bitstream generating process failures. For example, performance of a design improves with frequency until the memory bandwidth becomes a bottleneck. Function f can have varied degree of smoothness across dimensions and axis depending on the properties of the design. It will usually be bounded, as metrics like execution time or power both have to be positive. The noise included in $f(\mathbf{x})$ is due to system interaction. Summarizing, little is know about the function and hence, it is referred to as a noisy black-box function.

2.2.5 Optimization Challenges

Modern high performance computing world is facing the challenge of heterogeneous computing and increasingly complex designs. The designs are encapsulated in a far larger parameter spaces, have many possible architecture flavors and target various platforms while living in a range of different environments. Furthermore, with larger chips compilation and synthesis time is constantly increasing. The challenges faced by modern reconfigurable design developers are similar in nature to what has been approached by software developers in the past decades. How to offer the best performance while maintaining sufficient abstraction level. The main difference is that the parameter space for reconfigurable computing is orders of magnitude larger and more expensive to traverse - as a result the optimization process takes immense amount of time.

The are three major challenges. The main challenge in traversing the reconfigurable design parameter space is the long hardware generation and evaluation time. Although it

Table 2.4: Summary of the three major challenges with optimization of reconfigurable designs.

Challenge	Description
Hardware Generation Time	Generation and evaluation of a single parameter setting takes between an hour to two days.
No. of possible configurations	The curse of dimensionality. The number of potential design configurations can be in the millions.
Lost Knowledge	The knowledge gathered during development and optimization of designs is not transferred.

is possible to predict speed and area of a circuit before generating a bitstream, the process is inaccurate. Furthermore the timing of a circuit is unpredictable same as the output of place and route algorithms. The better the resource utilization obviously the better use of the existing hardware and usually better performance - unfortunately it is not always the case. Inserting an extra pipe might provide us with 10% percent improvement, but might degrade the achievable frequency by far more. It might make the place and route process orders of magnitude more time consuming. Currently no accurate models can encapsulate that. The current method of countering this phenomena is by tedious hand optimization and launching multiple PAR processes - a highly time consuming process.

After creating the design and determining target device, performance, energy usage, functionality and area utilization, the user has to optimize the design to meet the specified goal. A number of approaches have been investigated, some of which we presented earlier. Assuming that the average configuration space of a design lays in the range of tens or hundreds of different settings, each requiring many multi-hour simulations and builds, making exhaustive search of the parameter space infeasible. Current this problem is tackled by manual approach, yet they could be significantly improved and perhaps automated if more accurate and faster methods of parameter space exploration were employed.

Whenever an application is ported onto a heterogeneous system one can expect it to be used for some time, unfortunately at one point it is ought to be ported onto a next generation platform. Although the new heterogeneous system might have a similar architecture, this is not necessary the case. Introduction of new chip building blocks or a different intra node architecture might result in a new optimal design. This is a similar issue to software portability which is solved by various compiler and libraries, unfortunately in case of reconfigurable designs being leveraged by larger parameter space

and more expensive design evaluation.

2.3 Mathematical optimization

Mathematical optimization is understood as finding the best element from a set of available points [141, 41]. Often the element is a vector of scalars $\mathbf{x}$ defined over a multidimensional space $\mathcal{X}$. The fitness of the value is defined as a function f over the space. For maximization, this is stated as follows:

$$\underset{\mathbf{x}\in\mathcal{X}}{\operatorname{argmax}} f(\mathbf{x}). \tag{2.9}$$

A number of various optimization techniques can be employed. They are used depending on the properties of the function f and the search space $\mathcal{X}$. Convex programming can be used if the function is convex [105]. Convex programming is used for optimization of FPGA based designs [136]. Some other examples are optimization of an FPGA cluster, balancing computation and communication [95]. It was proven useful in communication systems and signal processing algorithms design [99]. Linear programming is a type of convex programming where the objective function is linear and the set of constraints are specified using only linear equalities and inequalities. A problem is a linear programming problem if it can be expressed as

$$\max \mathbf{c}^T\mathbf{x}, \text{ subject to } A\mathbf{x} = \mathbb{b} \geq 0 \text{ and } \mathbf{x} \geq 0. \tag{2.10}$$

where the problem has D variables, $\mathbf{c} \in \mathbb{R}^D$, $A \in \mathbb{R}^{D\times D}$ and $\mathbb{b} \in \mathbb{R}$.

Mixed integer linear programming are linear programming problems restricted to integer domains. They have been used for problems like heterogeneous multiprocessor system-on-chip design space exploration [51].

In geometric programming objective and inequality constraints are expressed as posynomials and equality constraints as monomials [40]. Some geometric programs can be expressed as convex programs. Some possible applications include optimization of integrated circuits [53] or design of FPGA architectures [137].

Nonlinear programming considers problems where some of the constraints or the objective function are non-linear [29, 85]. Non-linear problems are more common than

the linear ones, and the optimization is more complex. They are defined as

$$\underset{\mathbf{x}\in\mathcal{X}}{\operatorname{argmax}} f(\mathbf{x}). \tag{2.11}$$

subject to either one of k equality g or one of r inequality h constraints

$$g_i(\mathbf{x}) = 0 \quad i = 1..k. \qquad\qquad h_j(\mathbf{x}) \leq 0 \quad j = 1..r. \tag{2.12}$$

The techniques used to solve nonlinear programs are problem dependent, and the range can be considered unlimited. Some examples of nonlinear programming include image restoration [162] or centrifugal pump configuration [21]. Another example is a three heat-exchanger network synthesis problem [22]. Nonlinear programming has also applications in chemistry, as presented for a phase and chemical equilibrium problem [102]. It was shown useful for floor planning for FPGA designs [121].

Metaheuristics are algorithms that iteratively search the domain space of a problem to improve the quality of a solution, using some heuristic to assess quality [149]. Metaheuristics are favorable methods to deal with optimization problems whose search spaces for optimal solutions are extremely vast. One example of such algorithm is Particle Swarm Optimization (PSO). It is a population-based metaheuristic based on the simulation of the social behavior of birds within a flock [63]. The algorithm starts by randomly initializing a number particles where each individual is a point in the $\mathcal{X}$ search space. The population is updated in an iterative manner where each particle $\mathbf{x}_*$ is displaced based on the PSO particle motion equations. In the classical version for continuous $\mathcal{X} = \mathbb{R}^D$ spaces they are

$$r_1 \sim U(0,1), \qquad r_2 \sim U(0,1). \tag{2.13}$$

$$v_i = wv_i + c_1 r_1(l_i - x_i) + c_2 r_2(g_i - x_i). \tag{2.14}$$

$$x_i = x_i + v_i. \tag{2.15}$$

where i is dimension index, r_1 and r_2 are uniform random numbers, $\mathbf{l}_*$ is the particles $\mathbf{x}_*$ so far local best found position, $\mathbf{g}_*$ is the global so far best found position and $U(0,1)$ is a uniform random number with range $[0,1]$. The v_i are particle velocities, c_1 and c_2 are the acceleration coefficients and w is the inertia weight.

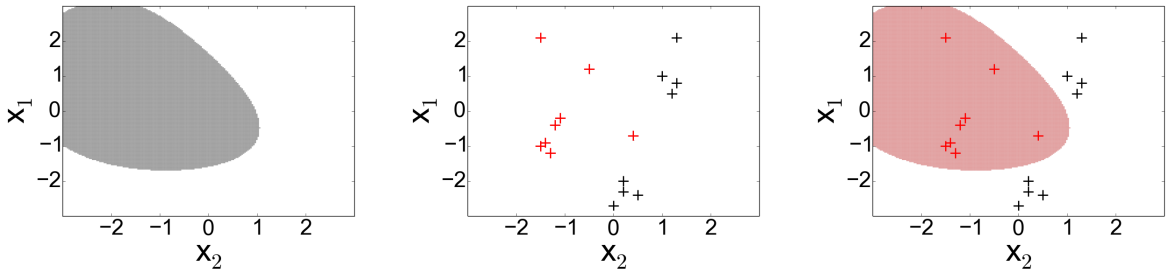
Metaheuristics have been extensively used across different domains, for example

Table 2.5: Mathematical Optimization Summary.

Approach	Details
Convex Programming	A problem of minimizing convex functions of convex functions over convex sets [105]. Linear programming is a major subtype of convex programming.
Nonlinear Programming	A problem of minimizing of nonlinear functions [85]. Number of problems is practically unlimited and solution is problem dependent.
Metaheuristics	Used when few or non assumptions can be made about the problem [149].

Genetic Algorithm (GA) have been used in aerodynamics to design new rotor-blades [67]. New algorithms are constantly being specified like Galaxy-based Search Algorithm (GbSA)[133, 134] offering various performance characteristics and finding new applications. Another example of a metaheuristic algorithm is Intelligent Water Drops (IWDP) [131, 132]. In [161] a new Ant Colony Optimization (ACO) based method is introduced for FPGA placement. It is compared to metaheuristics such as Simulated Annealing (SA), GA, PSO and hybrid GA and SA algorithm. Metaheuristics can be used to decrease data transfers by evolving clever compressions schemes while minimizing accuracy [125][126], automating optimizations within a hardware compiler or identifying reusable code sequences between the cores for multi-pass kernels.

The choice of the appropriate optimization approach is based on the problem. There are a number of important fitness function properties, which need to be taken into account. The problem might be multi-modal having multiple solutions that meet the target goal. The fitness function can be stochastic in nature. For example, the benchmark used to evaluate a design is based on random numbers and there is a certain noise associated with its output. To evaluate a reconfigurable design it first has to be synthesized, if this process fails the fitness is undefined resulting in f discontinuities. This is also a largely non-deterministic problem. The algorithm should be able to cope with mixtures of the previously mentioned parameter spaces. Generally, from the above mentioned only metaheuristics are applicable to the optimization of black-box functions. Unfortunately, they rely on a large number of heuristics evaluations and render this approach not applicable to reconfigurable designs. Evaluation of hundreds, and even less so thousands, of design parameter configurations is simply unrealistic. This makes creation of a generic



(a) *Ground Truth* (b) *Samples* (b) *Classification*

Figure 2.12: Classification example.

optimization tool for reconfigurable designs challenging.

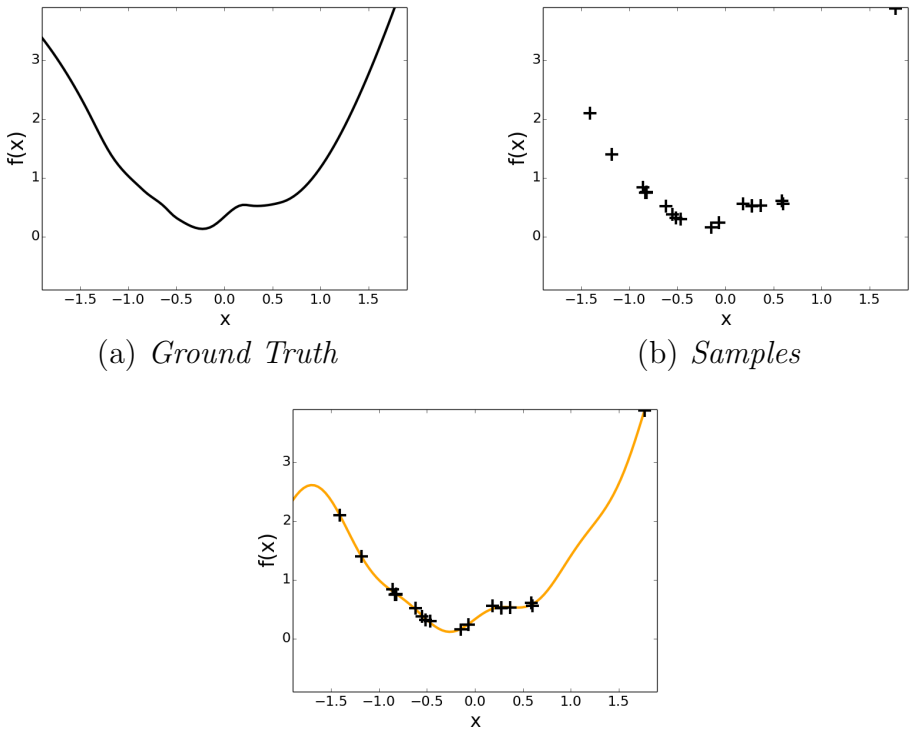
2.4 Machine Learning

Machine learning techniques are in essence a way of encapsulating knowledge in various forms, and then using it to achieve certain goals. For example it is possible to have a set of data, and for the algorithm to find structure within it. Another possibility, is for an algorithm to learn on which actions are most beneficial in certain situations. Yet, the type of algorithms relevant to this work do something else. They perform supervised learning, where the algorithm learns how a function works provided the input and output data.

2.4.1 Supervised Learning

In the simplest form there is a function $f(\mathbf{x})$ which takes an input variable and returns some output. The supervised learning algorithm infers this function from labeled observations that contain a number of input-output pairs. The problem is divided into regression when trying to determine the strength of a relationship between variables (e.g. forecasting of stock prices), or classification when the goal is to categorize inputs into distinct classes (e.g. learning to categorize faces in photos according to emotions they express). Examples of regression and classification are presented in Figure 2.13 and Figure 2.12. Among many others, supervised learning includes random forest, decision trees, nearest neighbors, probabilistic generative models, Artificial Neural Networks (ANNs), Support Vector Machines (SVMs), or Gaussian Processes (GPs).

An algorithm has a training phase. The user supplies the algorithm with a training set and the algorithm adjusts the function parameters so that it performs best according to some metric. If the model used for the function is powerful enough to represent the



(a) *Ground Truth* (b) *Samples*

(c) *Regression*

Figure 2.13: Regression example.

problem, and the training set was large enough, it should be able to generalize over the set of problems that it aims to solve [35]. As an example, it is impossible to accurately model a quadratic curve using a linear model (but for the most trivial case). It is possible to model it over a certain interval, and with limited accuracy. The task of the user is to choose the suitable model. Cost of the model, cost of training examples and the accuracy of output have to be taken into account.

This class of machine learning algorithms has been extensively studied as performance predictors for new processor micro-architectures, especially for soft-processors. Christophe Dubach proposes the use of machine-learning techniques to address architecture/compiler co-design [57]. In his work he shows how ANN can aid in processor and compiler co-design space exploration. This technique offers cheap design space exploration allowing to efficiently achieve the desired performance metric goal. In [48] authors investigate ANN as a micro-architectures soft error vulnerability predictor. They show how ANN can be reduce the cost of the design exploration to quickly access soft error susceptibility of a given hardware configuration.

Kernel Methods

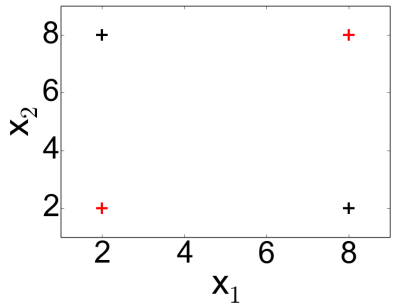
Kernel methods are a class machine learning algorithms, which make use of kernel functions. The idea is to solve a problem in a higher dimensional feature space, where often the solution is linear [35]. The problem is mapped to a higher dimensional feature space $\mathcal{X}_f$ using a feature map $\phi : \mathcal{X} \to \mathcal{X}_f$. Then, the problem is solved using a linear model. For example, in the case of classification, different classes can be separated by a hyperplane. The concept is presented in Figure 2.14.

Due to higher dimensionality one would rightfully expect the computational cost to increase. The kernel functions comes in handy by preventing that. When an algorithm relies on dot product computation, a cheap to compute kernel function is used to calculate the dot product in the higher dimensional feature space $\mathcal{X}_f$. This way, the problem can be solved in the higher dimensional space allowing for better predictive power without substantially increased computation cost. There are certain restrictions on the kernel function, yet they are not crucial in understanding of the kernel trick.

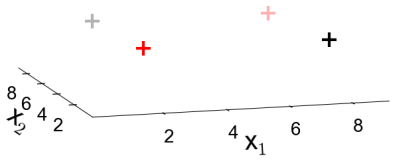
Support Vector Machines Classification

SVM is a maximum margin classifier, which constructs a hyperplane used for classification (or regression) [35]. SVM use kernel functions $k(\mathbf{x}, \mathbf{x}')$ to transform the input space to a different feature space where the data points are linearly separable. SVM are a class of decision machines and so do not provide posterior probabilities. There are n observations in a training set $\mathbb{X} = \{\mathbf{x}_i\}_1^n$ and $\mathbb{t} = \{t_i\}_1^n$, where $\mathbf{x}_i$ denotes an input vector, t_i denotes a target value. The column vector inputs for all n cases are aggregated in the $n \times D$ design matrix $\mathbb{X}$, and the targets in the integer vector $\mathbb{t}$. The goal is to classify an unseen input $\mathbf{x}_*$ based on $\mathbb{X}$ and $\mathbb{t}$ by computing a decision function d allowing prediction of a class label $d(\mathbf{x}_*) = t$.

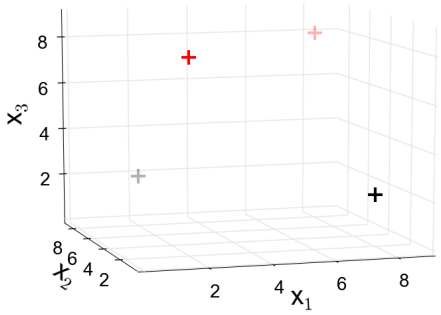
In SVM the decision function is constructed based on the observed data $\mathbb{X}$ and the target labels $\mathbb{t}$. SVM work by construction of a maximum margin hyperplane separating different classes. The problem is formulated for binary classification problem, with $t \in \{-1, 1\}$, although it can be easily extended to multiple class problems using a "one-against-one" approach [112]. In SVMs the decision function is constructed based on the observed data $\mathbb{X}$ and the target labels $\mathbb{t}$. SVMs work by construction of a maximum margin hyperplane separating the two classes



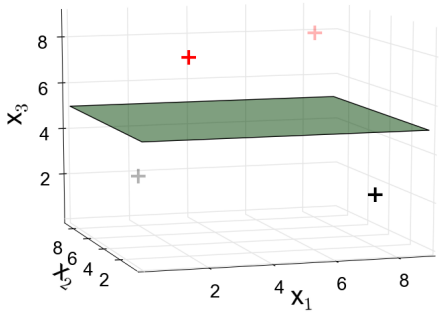
(a) *Non-linearly separable*



(b) *Non-linearly separable*



(c) *Same problem in the feature space*



(d) *Hyperplane separating points*

Figure 2.14: Feature space mapping and SVM classification. The two sets of points presented in (a) and (b) are not linearly seperable. Yet, when the problem is brought into a 3 dimensional feature space (c), the solution becomes apparent (d) and a maximum margin hyperplane is constructed.

$$\mathbf{w} \cdot \mathbf{x} - \mathbf{b} = 0. \tag{2.16}$$

where the hyperplane is defined using the Hessian normal form with $\mathbf{w}$ vector being the normal vector and $\mathbf{b}$ the distance from the origin. The objective function used to find the maximum margin hyper plane is

$$\underset{\mathbf{x},\mathbf{b}}{\operatorname{argmin}} \frac{1}{2}||\mathbf{w}||^2. \tag{2.17}$$

The objective function is subject to constraints $t_i(\mathbf{w} \cdot \mathbf{x}_i - \mathbf{b}) \geq 1$. The efficiency of

SVM technology comes from the idea of a support vector, only a subset ofnn parameter configurations of the observed data $\mathbb{X}$ and $\mathbb{t}$ is required to construct the hyperplane. Those nn parameter configurations, for which $w_i \neq 0$, are called support vectors. The complexity of SVM training is $\mathcal{O}(n^3)$, and of prediction it is $\mathcal{O}(nn^2)$ [115]. Using Lagrange multipliers α the objective function (2.17) is expressed as

$$\tilde{L}(\alpha) = \sum_{i=1}^{n} \alpha_i - \tfrac{1}{2} \sum_{i,j} \alpha_i \alpha_j t_i t_j \mathbf{x}_i^T \mathbf{x}_j. \tag{2.18}$$

SVMs become non-linear classifiers by using the kernel trick. The kernel trick is based on solving the problem in a feature space, which often has higher dimensionality. This can possibly allow for a linear solution to the problem, which does not exists in the original space. The transformation function ϕ is used to transform the vector $\mathbf{x}$ into the feature space. Now, by setting $\mathbf{x} = \phi(\mathbf{x})$

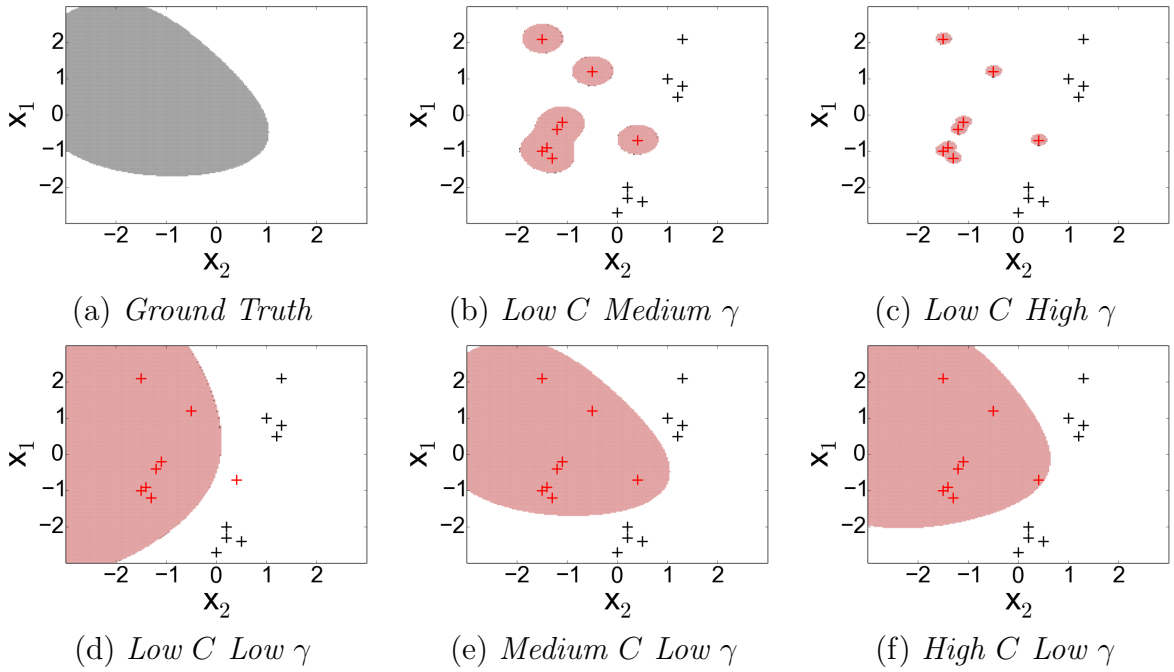
$$\tilde{L}(\alpha) = \sum_{i=1}^{n} \alpha_i - \tfrac{1}{2} \sum_{i,j} \alpha_i \alpha_j t_i t_j \phi(\mathbf{x_i})^T \phi(\mathbf{x}_j) \tag{2.19}$$

$$= \sum_{i=1}^{n} \alpha_i - \tfrac{1}{2} \sum_{i,i} \alpha_i \alpha_j t_i t_j k(\mathbf{x}_i, \mathbf{x}_j). \tag{2.20}$$

The kernel function replace the dot product $\mathbf{x}_i^T \mathbf{x}_j$. There are many possible kernel functions, squared exponential kernel with parameter γ being usually chosen. The parameter has to directly learned from the data, for example using cross-validation. Intuitively it can be understood as determining the sphere of influence of individual vectors.

$$k(\mathbf{x}, \mathbf{x}') = \exp(-\gamma ||x - x'||^2). \tag{2.21}$$

The SVM can be extended with a soft-margin hyperparameter C [50] if the data is not linearly separable. The soft-margin hyperparameter determines the penalty for misclassification of training samples [31, 35]. It enables construction of a decision boundary, which miss-classifies some parameter configurations, allowing the SVM to deal with a degree of noise during the classification process. Training of soft margin SVMs is done by optimization of the revised objective function. The new objective function introduces non-negative slack variables ξ_i, which allow for a degree of miss-classification. For a linear



(a) *Ground Truth* (b) *Low C Medium* γ (c) *Low C High* γ

(d) *Low C Low* γ (e) *Medium C Low* γ (f) *High C Low* γ

Figure 2.15: Two class SVM classification using different values of C and γ hyperparameters.

penalty function, the optimization problem becomes

$$\underset{\mathbf{x},\xi,\mathbf{b}}{\operatorname{argmin}} \tfrac{1}{2}||\mathbf{w}||^2 + C\sum_{i=1}^{n}\xi_i. \tag{2.22}$$

The constrains are modified to $t_i(\mathbf{w}\cdot\phi(\mathbf{x}_i)+\mathbf{b}) \geq 1-\xi_i$. The soft-margin SVM with a squared exponential kernel has two parameters C and γ, which are typically learned from the data by using cross-validation. Although the SVM is formulated for a binary classification problem, with $t \in \{-1, 1\}$, it can be extended to multiple class problems using a "one-against-one" approach [112].

The hyper-paramter C is less of a concern if γ matches the complexity of the problem [31]. Assuming that the real decision boundary is smooth and regular as presented in Figure 2.15(a), over-fitted model is seen in Figure 2.15(b) and Figure 2.15(c) due to high γ and overly flexible class predictions. Models presented in Figs. 2.15(d)-2.15(f) use low γ and offer good prediction mainly due to smooth decision boundaries and lower flexibility. The hyperparameters can be determined using cross-validation technique.

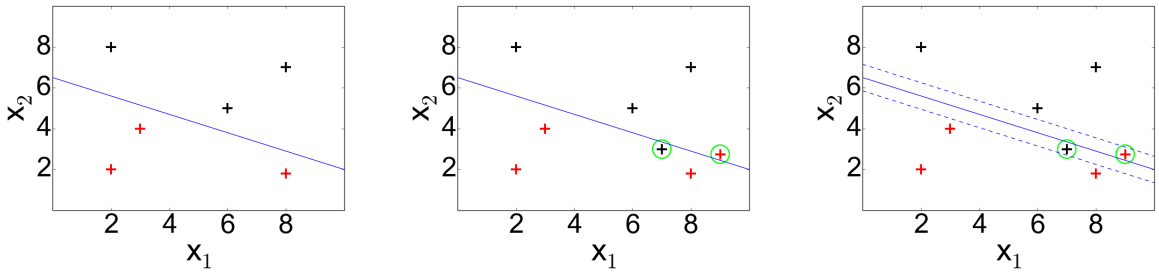


Figure 2.16: Soft margin SVM classification. A hard margin classification is shown in (a), where a linear decision boundary is counctructed in the feature space to seperate two classes. In (b), new examples are added and the problem is no longer linearly seperable. To take into account the noisy examples, or fact that the the classes are not linearly seperable in the feature space, a soft margin is introduced. In (c) this is represented by the two blue lines, where the SVM classifier is trained to allow for certain degree of misclassifcation.

Gaussian Process Regression

GP based regression models have been extensively used for Bayesian optimization. GP is a supervised learning method often used for probabilistic regression [130, 124]. It is based on a linear regression model with additive Gaussian noise.

$$f(\mathbf{x}) = \mathbf{x}^T\mathbf{w}, \qquad y = f(\mathbf{x}) + \epsilon, \epsilon \sim \mathcal{N}(0, \sigma_n). \tag{2.23}$$

where $\mathbf{x}$ is the input vector, weights $\mathbf{w}$ are the weights of the linear model, f is the modelled function and y is the observed value. The assumption is that the observed values include zero-mean Gaussian noise with variance σ_n. The presented model can be used to perform linear Bayesian regression, maximizing the probability of the observed data given the model and the model's parameters given the observed data $\mathbb{X} = \{\mathbf{x}_i\}_1^n$ and $\mathbb{y} = \{y_i\}_1^n$. Using the Bayes rule

$$\text{posterior} = \frac{\text{likelihood} \times \text{prior}}{\text{mariginal likelihood}}, \qquad \mathrm{p}(\mathbf{w}|\mathbb{y}, \mathbb{X}) = \frac{\mathrm{p}(\mathbb{y}|\mathbf{w}, \mathbb{X})\mathrm{p}(\mathbf{w})}{\mathrm{p}(\mathbb{y}|\mathbb{X})}. \tag{2.24}$$

Omitting the derivation, the posterior distribution becomes

$$\mathrm{p}(\mathbf{w}|\mathbb{X},\mathbb{y}) \sim \mathcal{N}(\tfrac{1}{\sigma_n^2}A^{-1}\mathbb{X}\mathbb{y},\ A^{-1}). \tag{2.25}$$

where Σ_p is the Gaussian prior covariance matrix of the weights $\mathbf{w}$ and the matrix $A = \sigma_n^{-2}(\sigma_n^{-2}\mathbb{X}\mathbb{X}^T + \Sigma_p^{-1})$. The prediction of $f(\mathbf{x}_*)$ for an input $\mathbf{x}_*$

$$\mathrm{p}(f|\mathbf{x}_*,\mathbb{X},\mathbb{y}) = \mathcal{N}(\tfrac{1}{\sigma_n^2}\mathbf{x}_*A^{-1}\mathbb{y},\ \mathbf{x}_*^TA^{-1}\mathbf{x}_*). \tag{2.26}$$

Then, the Gaussian process is a collection of random variables, for which any finite set is jointly Gaussian distributed. The goal in Bayesian regression is to obtain the predictive distribution $\mathrm{p}(f|\mathbf{x}_*,\mathbb{X},\mathbb{y})$ of the f function for a test input $\mathbf{x}_*$ given the observed data. This distribution can be viewed as a distribution over function regressions, as presented in Figure 2.17. The uncertainty of predication encapsulated in the form of a distribution, can be utilized in optimization. This contrasts with non-probabilistic regression techniques like simple linear regression or more sophisticated spline,which do not provide prediction uncertainty.

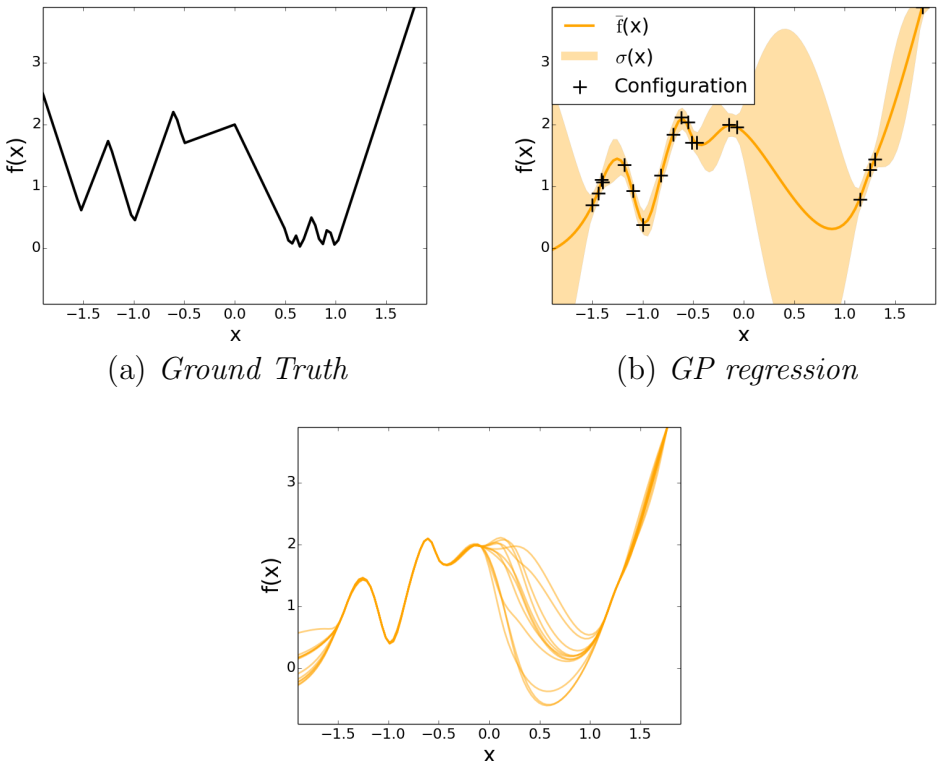
In GP, kernel functions are used to transform the original space to a feature space using transformation function ϕ, possibly yielding better predictive power. The previously described model (2.23) becomes

$$f(\mathbf{x}) = \phi(\mathbf{x})^T\mathbf{w}. \tag{2.27}$$

the predictive distribution becomes

$$f|\mathbf{x}_*,\mathbb{X},\mathbb{y} \sim \mathcal{N}(\tfrac{1}{\sigma_n^2}\phi(\mathbf{x}_*)A^{-1}\Phi\mathbb{y},\ \phi(\mathbf{x}_*)^TA^{-1}\phi(\mathbf{x}_*)). \tag{2.28}$$

where $\Phi = \Phi(\mathbb{X})$ is the aggregation of columns of $\phi(\mathbf{x})$ for all $\mathbf{x} \in \mathbb{X}$. Yet, introducing possibly higher dimensional feature space, the inversion of matrix A of size $N \times N$ in the (2.28) can become cumbersome. Ideally, this operation would be avoided. To achieve that, the equation (2.28) can be rewritten



(a) *Ground Truth* (b) *GP regression*

(c) *Sampling*

Figure 2.17: GP regression. A number of samples are drawn from the GP regression (b) and plotted in (c).

$$f|\mathbf{x}_*, \mathbb{X}, \mathrm{y} \sim \mathcal{N}(\phi_*^T \Sigma_p \Phi (\mathbf{K} + \sigma_n^2 \mathbf{I})^{-1} \mathrm{y},\ \phi_*^T \Sigma_p \phi_* - \phi_*^T \Sigma_p \Phi (\mathbf{K} + \sigma_n^2 \mathbf{I})^{-1} \Phi^T \Sigma_p \phi_*). \quad (2.29)$$

with $\phi(\mathbf{x}_*) = \phi_*$ and the $\mathbf{K} = \Phi^T \Sigma_p \Phi \in \mathbb{R}^{n \times n}$ is the kernel matrix. When an algorithm can be expressed solely by inner products in the input space, the inner product of transformation function $\phi(\mathbf{x})$ can be replaced by a kernel function. The computation is not directly performed in the higher dimensional space, instead a cheap-to-compute kernel function performs dot product in that higher dimensional space. This can drastically improve model expressiveness, with little extra compute cost. Seeing that the covariance matrix Σ_p is always positive semi-definite (it is a symmetric matrix with positive values), $(\Sigma_p^{1/2})^2 = \Sigma_p$. Then $\psi(\mathbf{x})$ is defined as $\psi(\mathbf{x}) = (\Sigma_p^{1/2})^2 \phi(\mathbf{x})$. Defining kernel function as $k(\mathbf{x}, \mathbf{x}') = \phi(\mathbf{x})^T \Sigma_p \phi(\mathbf{x})$ it becomes apparent that $\phi(\mathbf{x})^T \Sigma_p \phi(\mathbf{x}) = \psi(\mathbf{x}) \cdot \psi(\mathbf{x}')$. This is the kernel trick, which is crucial in GP technology. The computation is performed in higher dimensional while avoiding the problematic feature vectors $\phi(\mathbf{x})$. The kernel function

can also be viewed as the covariance between f evaluated at any two $\mathbf{x}$ and $\mathbf{x}'$ in the transformed feature space.

The GP does not require a predefined structure, and by using different kernels can approximate arbitrary function landscapes. GP based regression techniques have been used to solve highly dimensional problems in many fields such as robot control [106]. The GP are solely defined by their kernel functions $k(\mathbf{x}, \mathbf{x}')$ and the mean function $m(\mathbf{x})$. When using GP for regression, a prediction for an unseen input $\mathbf{x}_*$ is a Gaussian with mean and variance defined by kernel and its hyperparameters as well as the past observations $\mathbb{X}$ and $\mathbb{y}$. Without a loss of generality, it is common to normalize the data prior to training and prediction and to assume that the prior mean function is $m(\mathbf{x}) = 0$.

$$E[f(\mathbf{x}_*)] = \bar{f}(\mathbf{x}_*) = {\mathbf{k}_*}^T(\mathbf{K} + \sigma_n^2\mathbf{I})^{-1}\mathbb{y}. \tag{2.30}$$

$$\mathrm{Var}[f(\mathbf{x}_*)] = \sigma^2(\mathbf{x}_*) = k_{**} - {\mathbf{k}_*}^T(\mathbf{K} + \sigma_n^2\mathbf{I})^{-1}\mathbf{k}_*. \tag{2.31}$$

where $\mathbf{k}_* = k(\mathbb{X}, \mathbf{x}_*)$ and $k_{**} = k(\mathbf{x}_*, \mathbf{x}_*)$. When using GP for regression, the choice of a kernel function is critical for accurate modeling. The kernels themselves are specified with hyperparameters. A hyperparameter is a higher-order model parameter, which in the case of GP, are not learned from the data. The main features which should be taken into account when creating a GP regression model are isotropy, smoothness, and stationarity [124, 111, 35, 62].

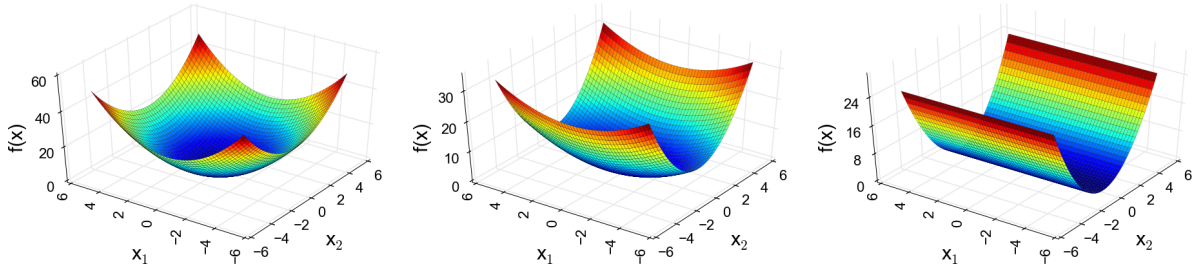


Figure 2.18: Anisotropic and isotropic functions. The image (a) represents a quadratic function in two dimensions: $f(\mathbf{x}) = {x_1}^2 + {x_2}^2$. This is an isotropic function, with equal impact of either of the dimensions. A different quadratic function is presented in (b), the function is anisotropic as the parameter x_1 has smaller impact on the function value than x_2. In (c) the situation is put to the extreme and the function solely depends on x_2.

Isotropy means uniformity in all orientations; all of the dimensions have similar impact on the function. This concept is presented in Figure 2.18. An anisotropic kernel function has different characteristics across the dimensions (orientations) defined by their respective hyperparameters. Kernels can be designed to automatically determine relevance of different dimensions using Automatic Relevance Determination (ARD) [35]. During the learning stage the algorithm determines the relevant hyperparameters. In case of GPs this is typically done by maximization of marginal log-likelihood. ARD is often used with the Gaussian squared exponential kernel

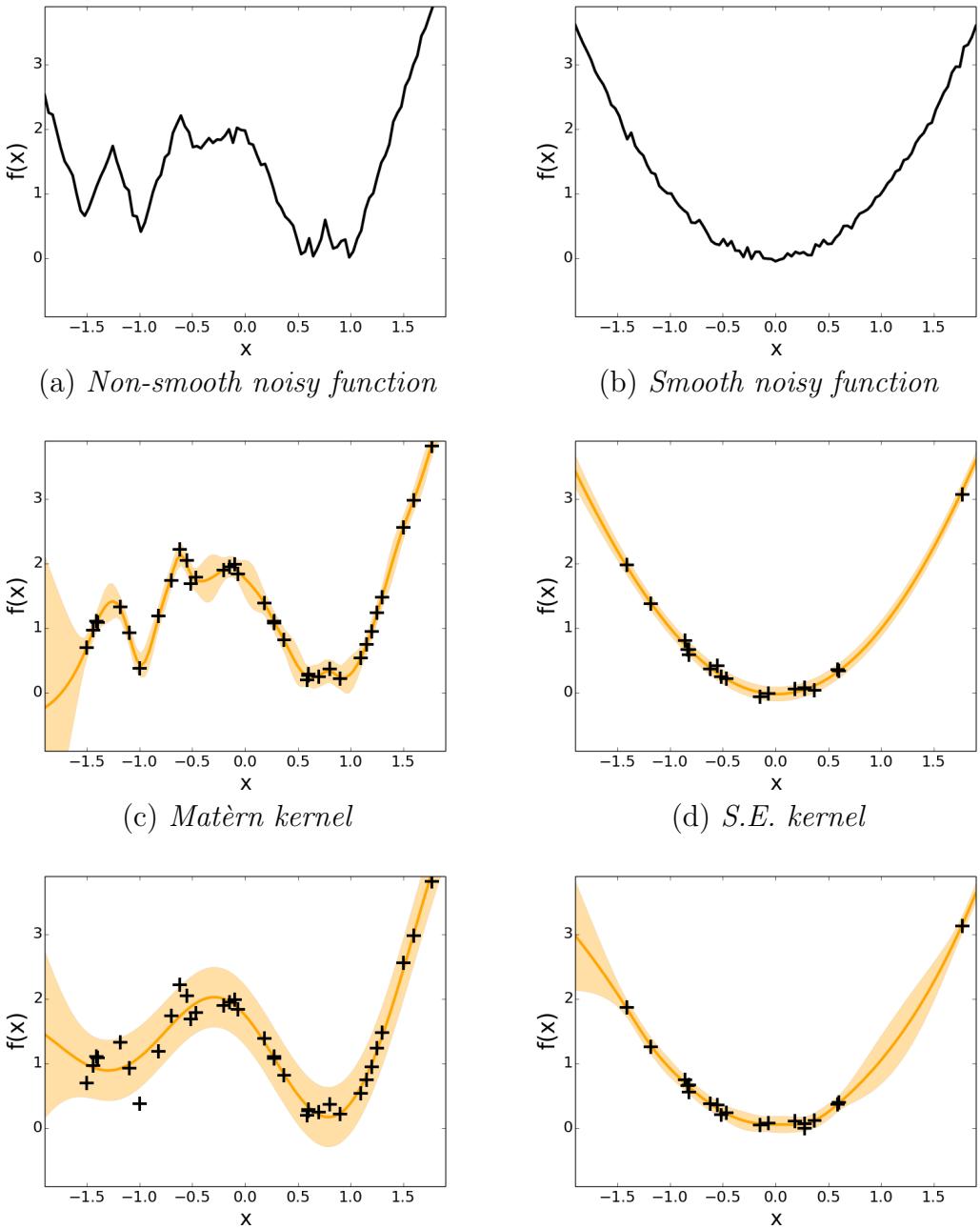
$$k(\mathbf{x}, \mathbf{x}') = \sigma_f^2 \exp(-\tfrac{1}{2}(\mathbf{x} - \mathbf{x}')^T \mathbf{M}(\mathbf{x} - \mathbf{x}'))). \tag{2.32}$$

with $\mathbf{M} = \text{diag}(|l_1^2, .., l_D^2|)$, where l_i is a length-scale hyperparameter defining the relevance of a parameter i and σ_f is the variance of the function f. The ARD comes here from the fact that during marginal log-likelihood maximization if the parameter i is not relevant l_i is going to be automatically minimized. The hyperparameter vector θ consists of all the length-scales and the noise variance hyperparameter δ. The kernel becomes isotropic when all of the length-scales are forced to be equal, giving an isotropic Gaussian squared exponential kernel

$$k(\mathbf{x}, \mathbf{x}') = \sigma_f^2 \exp(-\tfrac{1}{2l^2}||\mathbf{x} - \mathbf{x}'||^2)). \tag{2.33}$$

where l is the length-scale hyperparameter, and as previously mentioned σ_f is the variance of the function f and σ_ω is the noise variance.

Another important kernel function property is smoothness. It is intuitively understood as whether the function is slowly varying (smooth) or quickly varying (non-smooth). Kernel functions often have hyperparameters which dictate a degree of smoothness and will be set to a fixed value, depending on the expected properties of the function GP is meant to model. Examples of a noisy non-smooth and smooth functions are presented in Figure 2.19 (a) and (b). The non-smooth function is modelled quite accurately in (c) using a kernel with limited degree of smoothness. Equally, the smooth function is modelled accurately in (d). Their less accurate regressions are presented in (e) and (f), where worse matching kernel functions where used. The worst fit is easily seen by further deviated predicted mean, and much larger predicted uncertainty for (e) and (f) than (c) and (d). The commonly used Matèrn class kernel function



(a) *Non-smooth noisy function* (b) *Smooth noisy function*

(c) *Matèrn kernel* (d) *S.E. kernel*

(e) *S.E. kernel* (f) *Matèrn kernel*

Figure 2.19: GP regressions of non-smooth and a smooth function using Matèrn and squared exponential kernel functions. Figures (c) and (d) better modeled functions than (e) and (f), with clearly larger predicted process variance.

$$k(\mathbf{x}, \mathbf{x}') = \frac{2^{1-\nu}}{\Gamma(\nu)} \frac{\sqrt{2\nu}|\mathbf{x} - \mathbf{x}'|}{l}^{\nu} K_\nu \frac{\sqrt{2\nu}|\mathbf{x} - \mathbf{x}'|}{l}. \tag{2.34}$$

where l is the length-scale hyperparameter, K_ν is the modified Bessel function and ν hyperparameter regulates the degree of smoothness of the problem function f. The kernel in the above written form is isotropic, yet similarly to the squared exponential Gaussian kernel function, $\mathbf{M}$ matrix could be introduced. The ν hyperparameter regulates the degree of smoothness, the function $f(\mathbf{x})$ is differentiable k-times when $\nu > k$. Also, when $\nu \to \infty$ the kernel becomes the squared exponential kernel function and is infinitely differentiable. The hyperparameter ν is usually chosen to match the expected smoothness of the modeled problem, usually being set to 3/2 or 5/2. The kernel becomes:

$$k_{\nu=3/2}(\mathbf{x}, \mathbf{x}') = (1 + \frac{\sqrt{3}|\mathbf{x} - \mathbf{x}'|}{l})\exp\frac{-\sqrt{3}|\mathbf{x} - \mathbf{x}'|}{l}. \tag{2.35}$$

$$k_{\nu=5/2}(\mathbf{x}, \mathbf{x}') = (1 + \frac{\sqrt{5}|\mathbf{x} - \mathbf{x}'|}{l})\exp\frac{-\sqrt{5}|\mathbf{x} - \mathbf{x}'|}{l}. \tag{2.36}$$

The last major important kernel function property is stationarity. In mathematical terms, the stationary kernel function is solely a function of $|\mathbf{x} - \mathbf{x}'|$. Non-stationary kernels can model functions which completely change their behaviour in different areas of the space. A function can be quadratic in one region, and a sinusoid in another. As a GP with non-stationary kernel can generalize less across the space, it requires substantially more data for accurate modeling. Due to this fact, they are rarely employed.

There is a theoretical framework for obtaining the optimum hyperparameters [66] of a GP. The GP are usually trained by maximization of the log-marginal likelihood function with respect to the GP hyperparameters.

$$\log \mathrm{p}(\mathbb{y}|\mathbb{X}, \theta) = -\tfrac{1}{2}(\mathbb{y}^T(\mathbf{K} + \sigma_n^2\mathbf{I})^{-1}\mathbb{y} + \log|\mathbf{K} + \sigma_n^2\mathbf{I}|). \tag{2.37}$$

The problem of maximization of the likelihood function is non-convex. The complexity of training of a GP is $\mathcal{O}(n^3)$, it involves calculation of matrix inversion. Thanks to caching

of $(\mathbf{K} + \sigma_n^2\mathbf{I})^{-1}$, prediction of $\sigma^2(\mathbf{x}_*)$ scales quadratically $\mathcal{O}(N^2)$.

Cross-validation

It is crucial to choose the correct model and its hyperparameters [31, 35]. Cross-validation is one of the possible techniques. Cross-validation is a technique used to choose the most promising model, which involves splitting of the training data $\mathcal{D}$ into two subsets. One of the sets is used to train the model, while the other is used to validate it. Validation produces some goodness measure, which is used to assess quality of the model. This can be percentage of correctly predicted labels in the case of a classier. As an example, in the case of SVM with a Gaussian squared exponential kernel model is defined by its hyperparameter values γ and C. The technique could also be used to to compare SVM with different kernel functions.

The cross-validation algorithm is outlined in Figure 2.20. The training data is split several times, and the model is trained and evaluated using various training sets. The preferred model is one performing best across a number of different splits [35]. There are multiple schemes used for splitting of the training data into training and validation sets. This technique can often prevent model over-fitting, which is crucial, especially when the available training set has a small cardinality relative to the dimensionality of the problem.

```
Split 𝒟 into s training sets ⋃_{i=1}^{s} 𝒟_i;
for Each model do
    for i to s do
        Train the model using set 𝒟 − 𝒟_i;
        Evaluate the model using set 𝒟_i;
Return model with highest (lowest) score;
```

Figure 2.20: Cross-validation

2.5 Surrogate Models for Experimental Design

Using information feedback from a fitness function, usually the performance metric y and class label t, a surrogate model can be constructed and used by automatic optimization tool. A surrogate model is an approximation technique that is used when the behaviour of the underlying problem is not understood well enough to be treated analytically and is

expensive to measure, but at the same time the problem calls for automation. Properly chosen surrogate model can be trained to model the behaviour of the problem, and due to relatively low compute cost allow for automation. The benefit from using those models comes from the fact that they are orders of magnitude faster to evaluate than hardware generation and benchmarks, they can substantially accelerate optimization to enable an automated approach. The problem is that they need to be suitably chosen for the target problem.

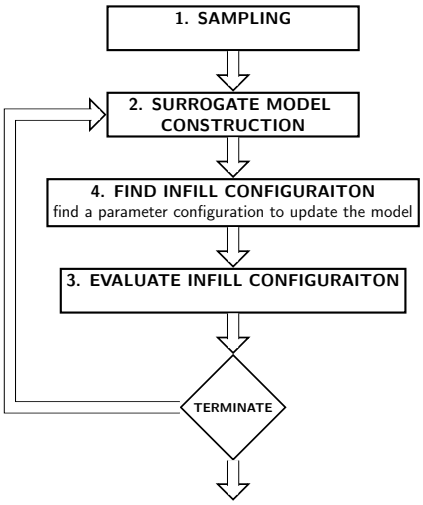
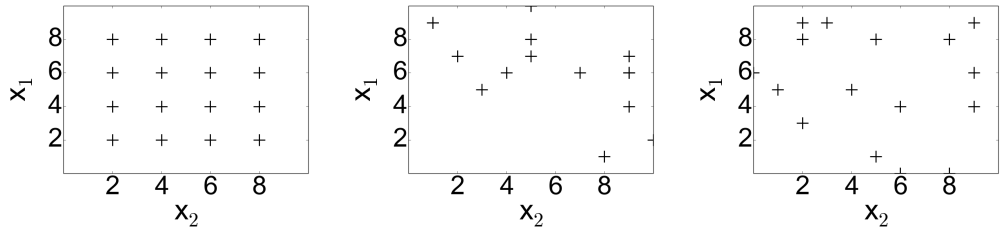


Figure 2.21: Surrogate Model Optimization

Surrogate models have been used in reconfigurable design optimization. Speeding-up time consuming high-level synthesis using surrogate modeling has been explored [117] using fitness inheritance. In [100] authors present a design and CAD tool parameter tuning approach. Equally to the design parameters, proper selection of those CAD tool parameters can allow for generation of better performing design. The technique offers a very high degree of parallelism, the authors generate up to 50 designs in parallel. Very few designers can build more than a few designs in parallel, let alone 10 or 50. Furthermore, the investigated case study requires only 20-25 minutes for hardware generation, an unrealistic example. Some of the proposed parameters, like random seeds used for placement, not necessarily being good candidates for optimization. Lastly, the authors claim that the presented approach offers a 20x reduction in the number of evaluated designs compared to exhaustive search. This makes the approach unrealistic for multi-parameter designs with millions of possible designs. Timing optimization using cloud computing and machine learning is presented in [80]. Again, the optimization targets CAD parameters. Multiple different designs are evaluated, and it is shown how tuning of the CAD parameters can

offer better chance of design meeting timing.

A typical surrogate-based optimization algorithm is presented in Figure 2.21. The algorithm starts with sampling of the parameter space. Sampling is necessary to construct an initial surrogate model. After evaluation of sampled configurations has finished, construction of the surrogate model, assessment of the next configuration to be evaluated and, finally, evaluation of the configurations follows in an iterative fashion. The configuration which is evaluated during every iteration is called infill: it can be understood as filling in the gaps in the understanding of the problem, different possible infill criteria exist [79, 144, 104].



(a) *Grid* (b) *Random* (c) *Latin Hypercube*

Figure 2.22: Examples of sampling plans.

Often a random sampling plan will be chosen; however, other plans like Latin hypercube plan [103] offer better space filling qualities which improve performance of the optimization algorithm [78]. Figure 2.22 presents three different sampling plans. A grid sampling plan would seem ideal, yet it struggles to deal with highly dimensional problems. In those cases the density of samples per space decreases . Random sampling plan has a tendency to oversample areas of space, while undersampling others. It is visible in Figure 2.22 where Random sampling plan oversamples the middle and leaves patches of empty space. The Latin hypercube sampling plan space filing properties are nearly as good as of the grid plan, but introduce randomness.

GP based surrogate models have been used for optimization of expensive fitness functions by many researchers [79, 78]. Usually, a surrogate model consists of a series of regressors, although for constrained optimization problems they can include classifiers, SVM in particular [89, 28]. In such a case, aside of modeling the fitness function the surrogate model determines which, if any, constraints are likely to fail. Extension of the surrogate model with a classifier allows for pruning of the parameter space: the classifier predicts which regions yield valid designs and thus prevents evaluation of unpromising configurations, which fail constraints such as being inaccurate or overmap on resources. This is crucial when parameter space is large and designs take a long time to evaluate.

2.5.1 Metaheuristics and Surrogate Models

Metaheuristics are algorithms that iteratively search the domain space of a problem to improve the quality of the solution [149]. They are a favorable methods to deal with optimization problems whose search spaces for optimal solutions are extremely vast. Although they can deal with discontinuous fitness functions, multimodal and even ones of stochastic nature, they rely on multiple evaluations of the functions. This becomes problematic when evaluation of the function is computationally expensive.

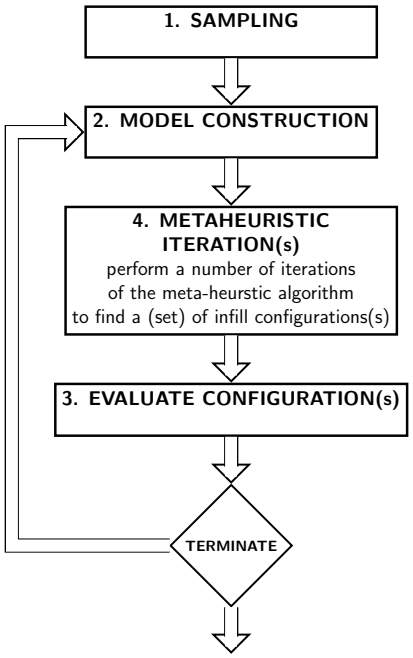


Figure 2.23: Meteahueristic Optimization and Surrogate Model

The commonly approached solution is integration of a surrogate model, which allows for an inexpensive alternative to function evaluation. An outline of metaheuristc algorithm with an integrated surrogate model is presented in Figure 2.23. The algorithm uses a metaheuristic to traverse the space, avoiding when possible function evaluation. Those function evaluations cannot be avoided all together, the algorithm has to use them to refine the model. This approach can be used to solve multidimensional problems [160]. Parallel algorithm with support for constraints is presented, an example of aerodynamic wing design is used for evaluation[163]. The algorithm offers parallelism for faster optimization. Different types of metaheuristics and surrogate model integrations are presented [34, 147]. In [44] an evolutionary algorithm based on GP is presented. A real example of a stationary gas turbine compressor profiles is evaluated. The problem with all of the presented

approaches is that they do not take into account non-deterministic and non-constant evaluation time of configurations $f(\mathbf{x})$. Furthermore, few, if any, realistic designs are used for evaluation. Lastly, the surrogate models do not learn about constraints boundaries.

As an example, the PSO algorithm can be extended to use a surrogate model, as presented in [66] based on a GP model or a Hybrid Surrogate Model (HSM) [150]. HSM is a hybrid surrogate model using a combination of Radial Basis Function (RBF) and quadratic approximation. The key step in integration of a surrogate model and a metaheuristic is introduction of a mechanism which determines when to evaluate the configuration and infill the surrogate. In the case of the GP based model the provided distribution allows for uncertainty assessment and subsequent configuration evaluation [66]. In the case of HSM, and generally models which do not allow for uncertainty assessment, different approach is needed. In [150] this is done by evaluating promising configurations found by the PSO algorithm. The PSO algorithm is complicated in its basic form [157], integration of a surrogate further aggravates it, not to mention the problem introduction of constraints.

2.5.2 Bayesian Optimization

Although automatic optimization is desired, it is data inefficient as it usually requires a lot of experiments. This is not possible in the case of reconfigurable computing, a single configuration takes hours to generate and for most designers the computing resources are limited and require a careful choice of experiments. Bayesian optimization is a powerful framework used when the configuration evaluation budget is small. It has been used extensively for experimental design – a systematic process where, through evaluation of an experiment, the cause and effect relationship of a design's input to its output is learned and optimized. It has been extensively used for optimization of black-box functions [79, 70]. Some of the possible applications include optimization of gait in robot locomotion [96], realistic image synthesis [60] or optimization of machine learning algorithms themselves [140]. In [104] an Gaussian Process – Upper Confidence Bound (GP-UCB) algorithm is used to help identify traffic congestion or to find locations of highest temperature in a building.

Bayesian optimization is based on a statistical surrogate model of the unknown performance function. The typically used GP surrogate model represents a distribution over possible functions f. The optimization starts with sampling of the parameter space, after which infill loop proceeds, as presented in Figure 2.24. During each iteration, a posterior distribution over performances is computed, accounting for all the previous

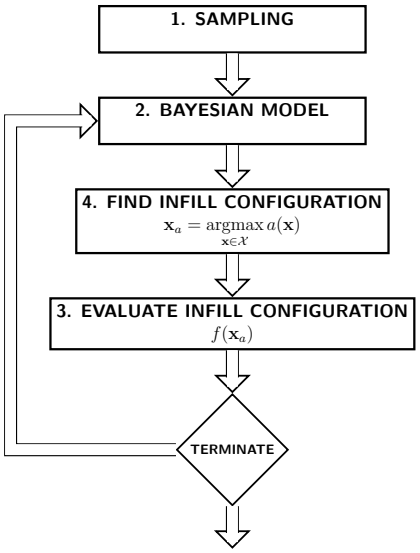


Figure 2.24: Bayesian Optimization

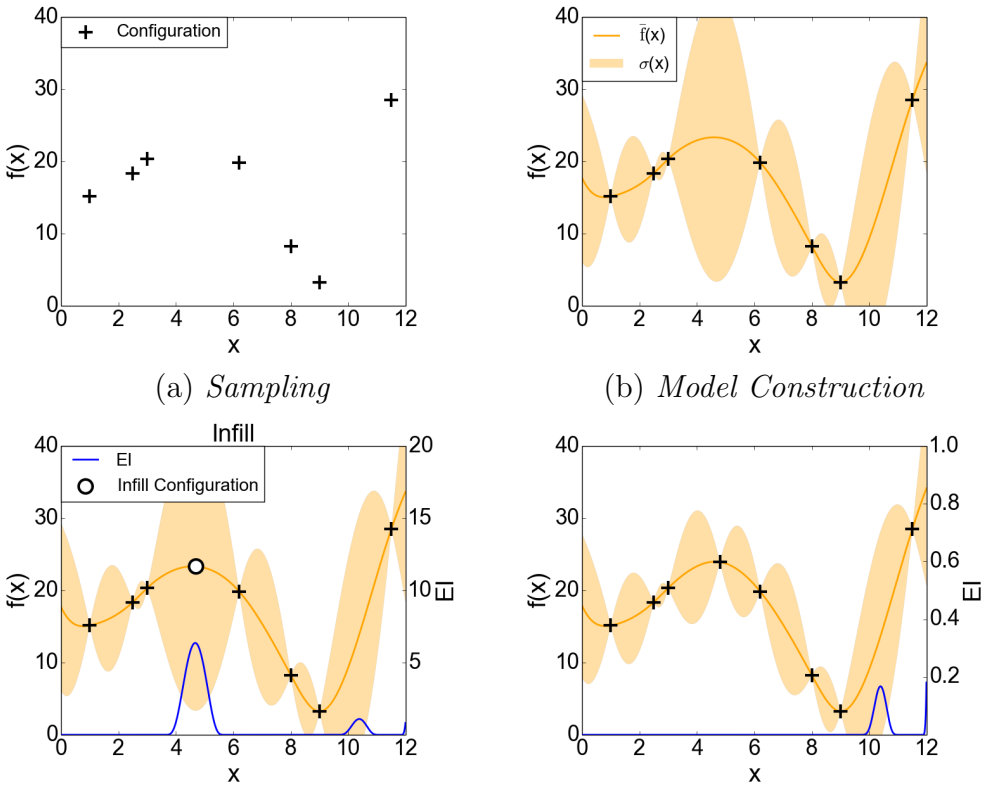
evaluations of the performance function. The user defines an acquisition function $a(\mathbf{x})$ which, expresses his belief on the desirability of points in the parameter space. This function, defined using the Bayesian model, is used to search for the next configuration to evaluate. This is done by maximization of an acquisition function over the parameter space $\mathcal{X}$ for unevaluated configurations

$$\underset{\mathbf{x}_*\in\mathcal{X}}{\operatorname{argmax}}\, a(\mathbf{x}_*). \tag{2.38}$$

The configuration is evaluated using the design fitness function f. Then the loop restarts with more data available for conditioning of the model. Bayesian optimization is beneficial when computation cost of the regression model and acquisition function maximization is smaller than evaluation of a configuration. Frameworks are available for multi-objective optimization problems [56]. Work has been extend to take into account constrained problems [28, 64]. To allow for parallel, and most importantly asynchronous, evaluation of configurations [75]. General frameworks are presented in [79, 97, 42].

Efficient Global Optimization

The EGO approach is a Bayesian optimization algorithm [79]. The key idea behind EGO is *data efficiency*, it only evaluate configurations that optimize its surrogate model. The algorithm is based around the Expected Improvement (EI) acquisition function, which



(a) *Sampling* (b) *Model Construction*

(c) *Infill* (d) *Design Evaluated*

Figure 2.25: EGO optimization. The algorithm starts with sampling after which it moves into the infill stage. First, the surrogate model is constructed and then used to calculate $E[I(\mathbf{x})]$. The configuration chosen for infill is the one offering the highest $E[I(\mathbf{x})]$. After configuration evaluation, the surrogate model is updated and the process continuous.

determines the expected improvement of a configuration over the best currently found configuration $\mathbf{x}^+$, s.t. $f(\mathbf{x}^+) = \max(\mathbb{y})$.

In particular, given the mean estimate $\bar{f}(\mathbf{x})$ and standard deviation $\sigma(\mathbf{x})$ by the GP regression, an improvement $I(\mathbf{x})$ over $\mathbf{x}^+$ is defined

$$I(\mathbf{x}) = (0, f(\mathbf{x}^+) - Y(\mathbf{x})). \tag{2.39}$$

$Y(\mathbf{x})$ is a Gaussian random number conditioned on the past observations $\mathbb{X}$, $\mathbb{y}$, i.e. $Y(\mathbf{x}) \sim \mathcal{N}(\bar{f}(\mathbf{x}), \sigma^2(\mathbf{x}))$, the distribution returned by the GP regression. The presented equation is defined for a maximization problem, such as throughput optimization. EI automatically manages exploration and exploitation by utilizing both the model uncertainty and fitness predictions. The $E[I(\mathbf{x})]$ of the improvement is

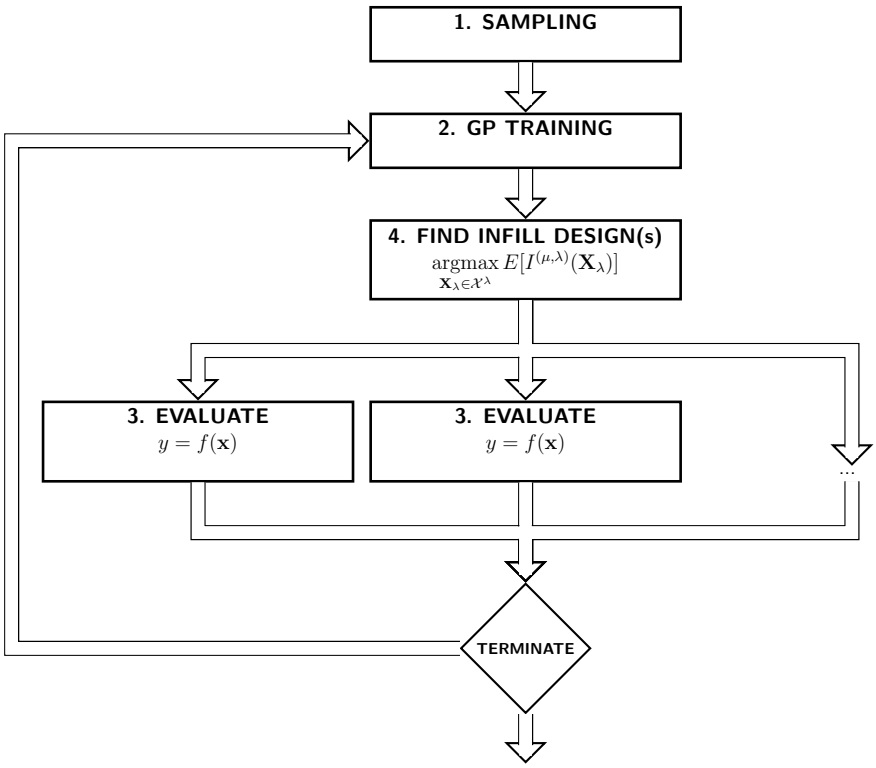


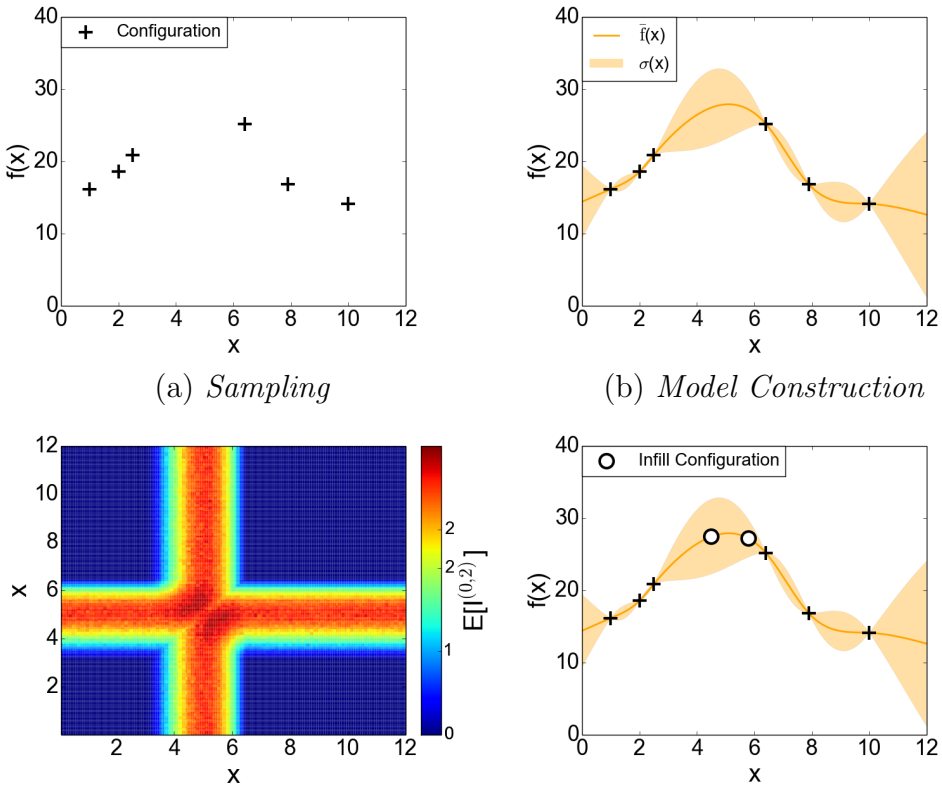
Figure 2.26: Asynchronous Parallel EGO Algorithm.

$$EI(\mathbf{x})] = f_\delta \Phi\left(\frac{f_\delta}{\sigma(\mathbf{x})}\right) + \sigma(\mathbf{x})\phi\left(\frac{f_\delta}{\sigma(\mathbf{x})}\right). \tag{2.40}$$

where minimization is desired; it can be modified for maximization problems. The symbols ϕ and Φ respectably denote the Gaussian distribution probability and cumulative density functions. To simplify notation $f_\delta = (f(\mathbf{x}^+) - \bar{f}(\mathbf{x}))$.

Asynchronous Parallel Efficient Global Optimization

EGO has been defined for parallel systems of P worker nodes [74, 75], based on new EI, $E[I^{(\mu,\lambda)}]$. At any given time μ nodes are busy and λ are idle. The goal is to find a set of configurations $\mathbf{X}_\lambda = \{\mathbf{x}_i\}_{i=1}^{\lambda}$ and to evaluate them using the idle nodes, while configurations $\mathbf{X}_\mu = \{\mathbf{x}_i\}_{i=\lambda}^{\lambda+\mu}$ are being evaluated on the busy nodes. For notation purposes all the points are aggregated in $\mathbf{X} = \{\mathbf{x}_i\}_{i=1}^{\lambda+\mu}$. The joint Gaussian vector $\mathbf{Y}^{\lambda+\mu} = \{Y(\mathbf{x}_i)\}_{i=1}^{\lambda}$ is conditioned on the past observations $\mathbb{X}$, $\mathbb{y}$ with covariance matrix $\Sigma = \text{var}(\mathbf{X}) = K(\mathbf{X},\mathbf{X}) - K(\mathbf{X},\mathbb{X})[K(\mathbb{X},\mathbb{X}) + \sigma_n I]^{-1}K(\mathbb{X},\mathbf{X})$. Having both predictions



(a) *Sampling* (b) *Model Construction*

(c) *EI Maximization* (d) *Infill*

Figure 2.27: Parallel EGO iteration with two worker nodes, the algorithm starts with sampling. Afterwards it moves into the infill stage. At first the surrogate model is constructed, with one configuration being evaluated. With one worker node free the current surrogate model is used to search for infill configuration by examining $E[I(\mathbf{x})]$. When a configuration is evaluated, the surrogate model is updated and new search begins.

the improvement criterion is defined:

$$I^{(\mu,\lambda)}(\mathbf{X}_\lambda) = \max(0, \max(\mathbf{Y}^\lambda) - \max(f(\mathbf{x}^+), \mathbf{Y}^\mu)). \tag{2.41}$$

The key to asynchronous parallel EGO is its efficiency even if the time taken to evaluate f for different configurations is non-uniform. This is common in reconfigurable designs, where depending on the design configuration the hardware generation time can vary from an hour up two two days. Parallel algorithms, which require synchronization at some step would need to wait for the configuration with longest evaluation time to finish. Visualization of the algorithm's iteration is presented in Figure 2.27.

Although the algorithm is promising, it suffers from two problems. A closed form solution to $E[I^{(\mu,\lambda)}]$ exists only for certain λ and μ values, in general it has to be estimated using numerical procedures, such as computationally expensive Monte Carlo methods [74, 75]. The second problem results from the curse of dimensionality. This is a multi-dimensional integral calculation, where the number of dimensions is $\lambda \times D$ and it depends on the number of idle nodes and the size of the parameter space.

$$\underset{\mathbf{x}_\lambda \in \mathcal{X}^\lambda}{\operatorname{argmax}} E[I^{(\mu,\lambda)}(\mathbf{X}_\lambda)]. \tag{2.42}$$

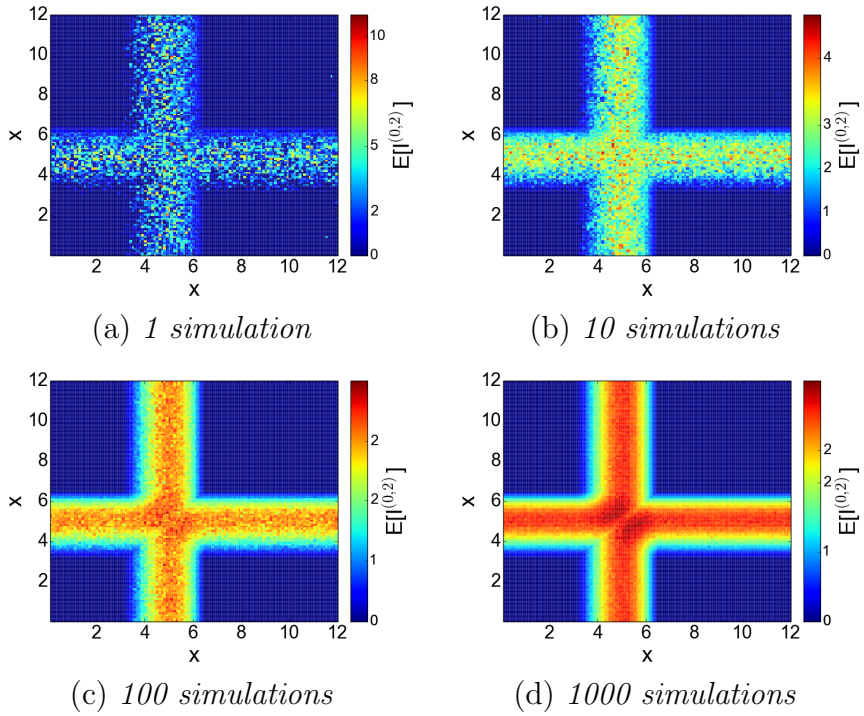
To decrease computational burden it is possible to compute bounds and devise an adaptive simulation management scheme [74, 75], yet even that might not be sufficient to discriminate alternative choices. The $E[I^{(\mu,\lambda)}]$ estimation procedure is presented in Figure 2.28. The problems resulting from Monte Carlo simulations can be clearly seen in Figure 2.29. Only at around 1000 simulations a clear peak is visible indicating a pair of promising configurations. Furthermore, maximization of $E[I^{(\mu,\lambda)}(\mathbf{X}_\lambda)]$ itself is non-convex, the problem is multi-modal and noisy due to reliance on the estimation of the objective function. Despite that, keeping the computational burden in check the algorithm offers asynchronous parallelization and a lot of promise.

$\Sigma = \text{cov}[f(\mathbf{X})] = K(\mathbf{X},\mathbf{X}) - K(\mathbf{X},\mathbb{X})(K(\mathbb{X},\mathbb{X}) + \sigma_n^2\mathbf{I})^{-1})K(\mathbb{X},\mathbf{X})$;
$\bar{f}(\mathbf{X}) = K(\mathbf{X},\mathbb{X})(K(\mathbb{X},\mathbb{X}) + \sigma_n^2\mathbf{I})^{-1})\mathbb{y}$;
L =cholesky(Σ) ;
for $i \in [0, 1, .., q]$ **do**
    `// Sample the Y vectors`
    $N \sim \mathcal{N}(0, \mathrm{I})$;
    $\mathbf{Y}^\mu, \mathbf{Y}^\lambda = \bar{f}(\mathbf{X}) + LN$;
    `// Calculate EI using sampled Ys`
    $ei \mathrel{+}= max[\max(\mathbf{Y}^\lambda) - \max(f(\mathbf{x}^+), \mathbf{Y}^\mu)]$;
output $\frac{ei}{q+1}$;

Figure 2.28: $E[I^{(\mu,\lambda)}(\mathbf{X}_\lambda)]$ estimation, as presented in [65] defined for a GP with a zero mean prior function.

Other Bayesian Optimization Algorithms

There are many other available Bayesian Optimization algorithms, based on a variety of different acquisition functions [140]. GP-UCB algorithm is presented in the context



(a) *1 simulation* (b) *10 simulations*

(c) *100 simulations* (d) *1000 simulations*

Figure 2.29: Varying number of simulatuions in $E[I^{(\mu,\lambda)}(\mathbf{X}_\lambda)]$ estimation. Although the ridge representing promising configurations is clearly visibile regardless of the number of simulations, the peak shown in image (d) can be difficult to localize. Note the symmetry.

of multi-armed bandit problems [26]. The multi-armed bandit problem is a technique where there is a number of alternatives to pick from, and the algorithm identifies the most promising one defined in terms of reward of a sequence of choices [81, 33]. This is typically defined using cumulative regret; the difference in the expected reward between the choices made and the best possible choice. This is different to global optimization, where the objective is to find the global optimum, not the sum of all previous best sampled function values. The multi-armed bandit problem takes its name from the problem of maximizing pay-off when playing slot machines in a casino. The player faces a problem, which machine and how many times to play. A slot machine is often called one armed-bandit, hence a row of them is called multi-armed bandits for gambling and gives a name to the optimization technique. A multi-armed bandit problem is one where user faces a number of alternative decision he has to make at each optimization step, and the goal is to maximize his reward. GP-UCB is based on a GP model, same as one used in EGO. The GP-UCB acquisition function is defined as

$$\underset{\mathbf{x}_* \in \mathcal{X}}{\operatorname{argmax}} [\bar{f}(\mathbf{x}_*) + \alpha_t^{1/2} \sigma(\mathbf{x}_*)]. \tag{2.43}$$

where α_t us domain-specific time-varying parameter, which regulates the trade-off between exploration through maximization of predictive mean and standard deviation. The user has to manage both the uncertainty of the GP model prediction, to explore poorly understood regions of the parameter space, and the mean prediction to exploit promising regions. In [144] authors compare GP-UCB with EI as well as Most Probable Improvement (MPI) [104] in the bandit setting, showing no significant difference between the first two. Yet, they present theoretical study on the convergence properties of the GP-UCB algorithm, something that was not that for EI or MPI. Batch parallelization of the algorithm is presented in [54].

The information-theoretic Predictive Entropy Search (PES) presented in [70] uses the expected information gain with respect to the global maximum as the acquisition function, advancement of similar concept presented in [158]. The acquisition function is designed to minimize the negative differential entropy of $\mathrm{p}(\mathbf{x}_{opt}|\mathbb{X}, \mathbb{y})$, or in other words maximize the information about the location of the global maximum. This is a different approach to EGO or GP-UCB which explicitly try to offset exploration based on the predictive posterior mean and uncertainty. A number of case studies are presented, including such as neural network training or optimization of the environment for optimal bacteria growth. According to authors the algorithm can offer benefit over EGO by being less greedy and using more experiments to explore the problem space.

Bayesian optimization is suited for constrained optimization [28, 156, 25]. In [28] authors extend EGO by integration of an SVM classifier. They evaluate two schemes of integration of probabilistic classifiers, first one is by discounting EI by the probability of $\mathbf{x}$ being valid

$$EI_c(\mathbf{x}) = \mathrm{p}(c(\mathbf{x}) = 1|\mathbb{X}, \mathbb{t})EI(\mathbf{x}). \tag{2.44}$$

similar to the probability adjusted EI formulated in [127]. The second method is derived by nullification of $EI(\mathbf{x})$ for configurations with probability higher than 50%.

$$EI_c(\mathbf{x}) = \begin{cases} EI(\mathbf{x}), & \mathrm{p}(c(\mathbf{x}) = 1|\mathbb{X}, \mathbb{t}) > 0.5 \\ 0, & otherwise \end{cases}$$

they use a modification of probabilistic SVM formulation as their classifier [118]. Only one artificial problem is used for evaluation, and the constrained formulation is shown to offer more consistent performance. An alternative approach is to use "expected violation" as presented in [25]. The "expected violation" is the expectation of the amount by which the improvement is violated and is used to discount EI.

2.6 Conclusion

The three major challenges associated with the development, and in particular parameter optimization, of reconfigurable desings are the result of reconfigurable hardware characteristics, customization and portability. First two challenges are large parameter spaces and long hardware generation time. The design parameter spaces often involve millions of possible design configurations. The size of the parameter space combined with long hardware generation time makes the optimization particularly challenging. Manual development techniques have been developed and successfully applied at multiple levels of reconfigurable design development, ranging from the development of specialized FPGA chip architectures up to domain-specific frameworks, yet always requiring substantial effort and high level of expertise from the designer. Often mathematical programming is used by the designers during the optimization stage, yet it requires careful study of the optimized design. This can require development of new design specific optimization tools. More generic metaheuristic algorithms can deal with large optimization spaces, yet they require multiple heuristic evaluations. Due to the high cost of hardware generation generic metaheuristic approach is unfeasible for reconfigurable hardware.

Yet, various techniques have been proposed to encapsulate design characteristics and minimize the cost involved in traversing the large parameter space. In particular, surrogate models for expensive optimization have been shown useful in many fields. The surrogate model based optimization involves modeling of the target problem using cheap to compute models offsetting expensive design evaluations. Those models are often based on machine learning techniques, mainly supervised learning in the form of regression and classification. Both surrogate model aided metaheuristics and Bayesian optimization show a lot of promise.

The third and last challenge is to create automated optimization algorithms which would efficiently traverse expensive parameter spaces, while learning from previous optimization. The information gathered during optimization of a design, can be often later transferred to yield faster and more accurate optimization of a new similar design. This is very common in reconfigurable computing, where designs are often ported across different platforms or customized for different goals like power efficiency or speed.

Chapter 3

Design by Particle Swarm Optimization

This chapter presents our attempt to optimize reconfigurable designs using surrogate modeling and metaheuristics, in particular PSO. Traditionally, optimization involves manual design analysis, modeling, and exploration tool creation. This requires an experienced designer, and is a time consuming process. We develop the MLO tool to automate this process. From a number of benchmark executions, we automatically derive the characteristics of the parameter space and create a surrogate model of a fitness function through regression and classification. Based on this surrogate model, design parameters are optimized using metaheuristics. The work is published in [91, 89].

On numerous occasions FPGA designs have been shown to offer power and speed advantages over pure software solutions, but at a high design effort. In particular, the designer has to describe the design using a hardware description language, validate and synthesize it. The design will often be defined with various parameters, allowing for optimization to match specific reconfigurable devices and design goals. Examples of such parameters are the number of computational engines, numerical representation, clock frequency, degree of pipelining. Determining the optimal configuration is difficult as the parameters often allow for many possible design parameter configurations, each taking large amounts of time to evaluate. Synthesis of a design and its subsequent testing can take many hours.

Systematically optimizing a design for several parameters requires the designer to analyze the design, create models and benchmarks, and subsequently use them to optimize the design for throughput, power consumption, or some other performance metric. This is a time consuming process, often lasting a number of days. Previous research has

considered optimization of parametric designs, such as optimization of multi-FPGA systems [71]. The level of parallelism can have non-obvious impact on the performance of run-time reconfigurable designs [30]. The design throughput and power [155, 46] are highly dependent on the numerical representation. It influences resource utilization and therefore the level of parallelism. Optimization of coefficients of constant multipliers can also yield improvement [76]. Balancing data-reuse and loop-level parallelism for hardware generation requires complicated frameworks [94]. Determining optimal stencil configuration is known to be a difficult problem [109]. The result of optimization is the optimal parameter setting and usually a set of tools and models which can be used to re-optimize the design for a different device. Although these design specific optimization tools and models might be fast to use, their development time is often long. Furthermore, the design specific optimization schemes have to be updated for new designs, the high manual effort and low re-usability of the tools making this approach highly unproductive.

Generic automated tools can be quickly deployed and require little development time, yet they often lack efficiency and optimization time will increase. Speeding-up expensive evaluations in high-level synthesis using surrogate modeling has been previously explored [117]. Fitness Inheritance was used to decrease the number of configuration evaluations and hence speed-up optimization. In previous work [91, 89] we have shown it is useful to construct surrogate models of fitness functions representing the design quality of reconfigurable parameterized designs. As these models are orders of magnitude faster to evaluate than generation of bitstreams and code execution of benchmarks, they can substantially accelerate optimization, enabling an automated generic approach.

This is the motivation behind our development of the MLO tool, which we apply to the problem of reconfigurable design parameter optimization. We are particularly interested in parameters with numerous possible settings over which the design's performance exhibits a clear correlation. The algorithm is inspired by [66], with a major and crucial difference of using a classifier to account for invalid regions of the parameter space. We investigate various MLO components particularly looking into the construction of GP surrogate models and their kernel functions to determine their impact on algorithm's performance. The novel aspects are:

- We combine surrogate models with metaheuristics to provide a new MLO algorithm for automated optimization of reconfigurable design parameters. It is based on a generic surrogate model, which approximates the reconfigurable design fitness function using regression and classification. (Section 3.2)

- A detailed evaluation of our MLO approach based on four case studies with variable precision and parallelism: (a) a quadrature-based financial design [91], (b) a PQ design [46], and two designs based on Sequential Monte Carlo (SMC) methods (c) a stochastic volatility design and (d) a robot localization design [47]. A careful study of algorithm's components is performed to assess its robustness. The evaluation includes the study of the algorithm's parameters impact on optimization performance. The algorithm can improve optimization time by up to 50% compared to design specific optimization tools. (Section 3.3)

3.1 MLO Optimization Approach

The MLO approach is an automated generic optimization approach as presented in Subsection 2.2.1. It aims to solve the reconfigurable design optimization problem presented in Section 2.2.2. The algorithm is used to optimize designs, as presented in the problem statement section. The great advantage of MLO is that it simplifies the optimization flow compared to manual and design specific approaches. The designer saves time on coding of the optimization tools and on building analytical models as glsalo is used instead. Guaranteed MLO offers comparable optimization time, this allows for improved designers productivity. The process is illustrated in Figure 3.1. It takes as an input hardware generation and evaluation scripts, parameter space and goal definitions. Sample input is presented in Figure 3.2. Those inputs are used by the algorithm to evaluate a number of designs and, through the construction of a surrogate model, find the optimal design parameter configuration.

The key idea behind MLO is use of the PSO algorithm extended with a surrogate model. Metaheuristics are used to explore the parameter space and find the optimal configuration. The challenge in applying metaheuristics to reconfigurable design parameter optimization is related to the high cost of fitness function evaluations (design benchmark evaluations involving bitstream generations). The high cost is avoided by using the surrogate model, and occasionally evaluating configurations. The algorithm's convergence is improved by integration of an SVM classifier to prune the invalid regions of parameter space. The classifier is used to create a decision boundary to determine if the design would likely fail to meet certain constraints such as timing or accuracy; and those undesirable regions of the parameter space are avoided. This further improves optimization efficiency, avoiding unnecessary parameter configuration evaluations.

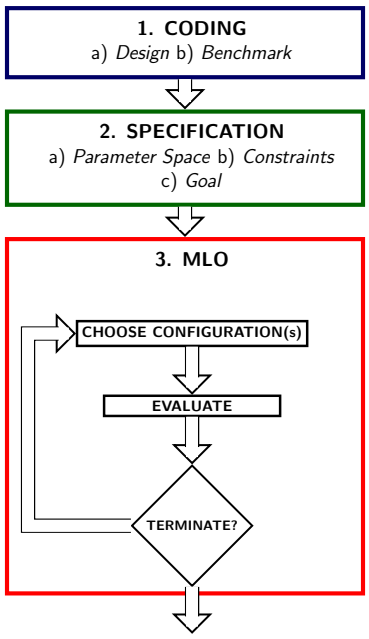


Figure 3.1: MLO optimization approach.

3.2 MLO Algorithm

The algorithm is designed to offload the process of optimization from the designer whist maintaining data efficiency. The algorithm works on the assumption that the designer prepared the required input. That is the definition of the parameter space $\mathcal{X}$, and a benchmark function b. The benchmark function evaluates fitness and the validity of a configuration $\mathbf{x}$. Typically, this is a script which invokes hardware generation and, once this is finished, executes a benchmarking program. That script, along with the parameter space definition, is used by MLO for optimization. A higher level overview of the algorithm is presented in Figure 3.3. It starts with the Latin hypercube sampling plan [103, 62]. The data collected during sampling is used to build the initial surrogate model. The model training starts with feature standardization. Afterwards, training of the GP and SVM proceeds. Lastly an iterative process follows using the PSO algorithm to find the optimal design configuration. After each iteration the model is rebuilt. The MLO algorithm has two unique features:

1. Integration of a classifier and a regressor for reconfigurable design modeling. This allows the algorithm to prune unfavorable regions of the parameter space. As the space is expensive to explore due to high hardware generation cost, the integration of the classifier is one of the main features, which enables automation.

```
# parameter space definition
parameters = {"freq_min" : 100,
              "freq_max" : 100,
              "p_min" : 1,
              "p_max"  : 10,
              ...
              }

# Build bitstreams and run benchmarks, the fitness function
def buildHardwareRunBenchmark():
      # execute bitstream generation
        os.system("Make hw")
      # execute benchmark and / or analyze bitstream
        return os.system("Make run")

# supply the parameter definition and scripts to the optimization algorithm
optimialDesign = MLO(parameters, buildHardwareRunBenchmark)
```

Figure 3.2: The input includes the parameter space specification, and scripts used to generate and benchmark bitstreams.

2. Integration of an infill scheme to aid PSO. The PSO algorithm on its own has complex dynamics. It's behavior gets more difficult to analyze with the integration of a surrogate model.

First, to the best of our knowledge, classifiers have not been integrated in surrogate model based algorithms for reconfigurable design optimization to decrease the search space size of reconfigurable designs. The concept was presented in [89, 91]. It is similar to the concept presented in [72] where the authors define simple, yet effective, constrained PSO algorithm. Their methodology is an extension of the PSO algorithm, which is not applicable to expensive optimization problems. There are some surrogate-based algorithms using a combination of a regressor and a classifier applied to constrained expensive optimization problems outside of reconfigurable computing world [25, 156, 28]. In particular, GPs and SVMs are used in [28]. They use probabilistic SVMs to modify the EI acquisition function. A similar concept defined for asynchronous parallel optimization problem is presented in Chapter 4; meanwhile this chapter presents work on surrogate model aided PSO algorithm. The advantage of the concept of using a classifier to treat constraints is a decreased parameter space, which should allow for faster convergence of the algorithm. This is essential in the case of reconfigurable designs, as it limits the number of lengthy hardware generations. The drawback is increased complexity of the algorithm and problems which can arise. This is partially mitigated by using simple

concepts introduced in [72].

The probabilistic classifiers like Relevance Vector Machine (RVM)s [154] or GPs for classification [124], investigated for expensive global optimization in [28], are conceptually very attractive. However, their integration with a surrogate model can become problematic, especially in the context of metaheuristics. The predictive probability has to be used to either discount desirability of a configuration or a threshold minimum predictive probability has to be defined; configurations with probability lower than a certain value are not evaluated. The approach followed in this work is to use a non-Bayesian classifier and to identify the valid region $\mathcal{V}$ using SVMs. The goal is to identify the best possible design before exhausting time budget; with the budget being typically very limited main aim is to find configurations with high $\mathfrak{p}_{\mathbf{x}}(1)$ rather than exact $\mathcal{V}$. The belief is that exploring nearby area of valid observed designs is sufficient, task for which SVMs are the most adequate.

The second new unique feature is the integration of an infill scheme to aid PSO. If the GP model fails to build, or the standard deviation is too high for the metaheuristic to proceed, infill parameter configuration is evaluated with the highest predicted fitness. Thanks to the infill scheme the algorithm is simplified and offers better performance. Through the introduction of a classifier, discrete spaces and limitation of search space boundaries the algorithm can end in a critical state. For example, particles can collapse into a single point, which ceases any motion and therefore the algorithm becomes idle. Another potential problem is when all particles are placed outside of the valid area, again potentially ceasing all dynamics.

There is one limitation of the MLO algorithm. It cannot be used to optimize unstructured parameters. Those are the parameters for which no underlying meaningful distance metric exists. MLO relies on the fact that performance of design with different parameter configurations exhibits correlation across a parameter range, this is not the case for unstructured parameters. An example would be a parameter which defines which chip or memory controller is being used. The possible parameter spaces are defined in the following subsection.

3.2.1 Latin Hypercube Sampling

The algorithm uses $\mathfrak{n}$ particles aggregated in $X_* = \{\mathbf{x}_*\}^{\mathfrak{n}}$. Each particle has an associated fitness $\mathbf{x}_f$ and a position $\mathbf{x}$. A Latin hypercube plan offers good space filling qualities, which can improve performance of the optimization algorithm [103]. It relies on the

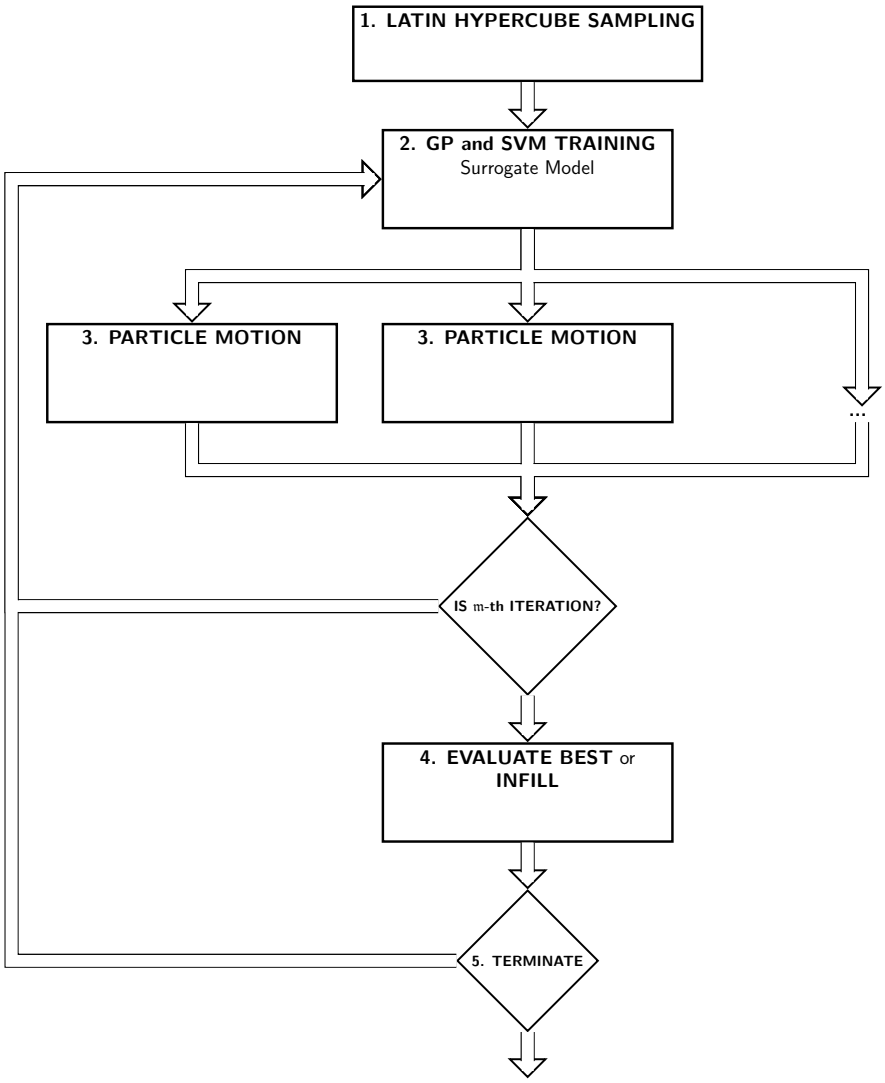


Figure 3.3: MLO Algorithm overview.

concept of Latin square. A sampling on a square grid is only a Latin square if each each row and column contains at least one sample. A Latin hypercube follows the same concept defined for hypercubes. The algorithm starts with the evaluation of a number of design parameter configurations equal to the number of particles, as indicated by the Latin hypercube sampling. Those are the starting positions of the particles. An illustration of sampling in one dimension is presented in Figure 3.4. Note that the evaluation of the design with one of the configurations failed, and is marked in red.

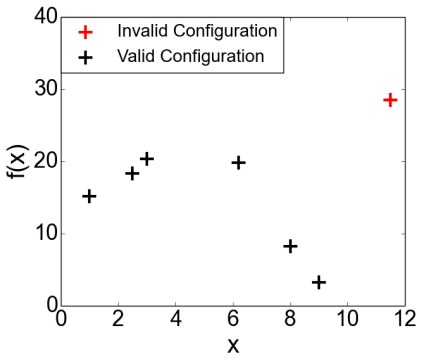


Figure 3.4: MLO iteration, the algorithm starts with Latin hypercube sampling.

1. Treat each of the D parameters as a continuous range of numbers. When sampling $\mathfrak{k}$ points, divide each parameter into $\mathfrak{k}$ equal sized intervals. This procedure forms a hypercube;
2. For each of the D parameters, sample using uniform distribution one point from each of the $\mathfrak{k}$ intervals;
3. Perform random permutation of the samples to form $\mathfrak{k}$ D-dimensional tuples. This forms a Latin hypercube;
4. Discretize the position of the particle so that they are valid in the parameter space $\mathcal{X}$ and evaluate them;

Figure 3.5: The procedure for Latin hypercube sampling on a discrete space.

3.2.2 GP and SVM Training

After sampling, and whenever new configurations are evaluated, the surrogate model is reconstructed. The training data $\mathbb{X}$ and $\mathbb{y}$ is standardized prior to both the GP and SVM training. The surrogate model consists of a GP regressor and SVM classifier. Defined for the purpose of GP training is the set of configurations that resulted in hardware generation and fitness evaluation $\mathbb{X}^+ = \{\mathbf{x}_i | \exists \mathbf{x}_i : [\mathbf{x}_i \in \mathbb{X}] \wedge [t_i \in \mathcal{T}^+]\}$. Given that set, associated design fitness observations $\mathbb{y} = \{y_i | \exists \mathbf{x}_i : [\mathbf{x}_i \in \mathbb{X}^+]\}$ and model hyperparameters θ the GP computes the predictive distribution $\mathrm{p}(f|\mathbf{x}_*, \mathbb{X}^+, \mathbb{y}, \theta)$ for new configurations. The SVM predicts the class $t = d(\mathbf{x}_*)$ for all configurations (i.e. regardless whether the configurations are valid or not) using all observations $\mathbb{X}$ and the observed target labels $\mathbb{t} = \{t_i\}_1^n$. The goal of classification is to construct a decision function d, which allows prediction of class labels $d(\mathbf{x}_*) = t_*$. An illustration of the surrogate model is presented in Figure 3.8.

$L = \text{cholesky}(K(\mathbb{X}^+, \mathbb{X}^+) + \sigma_n^2 \mathbf{I})^{-1})$;
$\mathbf{a} = L^T \setminus (L \setminus \mathbb{y})$;
$\log \text{p}(\mathbb{y}|\mathbb{X}^+) = -\frac{1}{2}\mathbb{y}^T\mathbf{a} - \Sigma_i L_{ii} - \frac{n}{2}\pi$;

Figure 3.6: Mariginal likelihood calculation for Gaussian process regression [124].

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 ARD [35] squared exponential kernel function with additive Gaussian noise, which allows learning of the impact of different parameters on the parameter space [124]. This kernel function has different characteristics across the dimensions (orientations) defined by their respective hyperparameters. The SVM uses soft-margins [50] to account for a degree of noise and the squared exponential kernel as smooth class boundaries are expected.

$\mathbf{v} = L \setminus \mathbf{k}_*^+$;
$\bar{f}(\mathbf{x}_*) = L \setminus \mathbf{k}_*^+$;
$\text{Var}[\bar{f}(\mathbf{x}_*)] = K(\mathbf{x}_*, \mathbf{x}_*) - \mathbf{v}^T\mathbf{v}$;

Figure 3.7: Prediction of mean $\bar{f}(\mathbf{x}_*)$ and of the variance $\text{Var}[\bar{f}(\mathbf{x}_*)]$ of the estimate using Gaussian process regression [124]. The matrix L and vector $\mathbf{v}$ are computed during model training. The vector $\mathbf{k}_*^+$ is the vector of covariances betwen the configuration and training set $k(\mathbb{X}^+, \mathbf{x}_*)$.

The training of the GP is performed by marginal log-likelihood maximization as presented in Figure 3.6. Due to non-convex nature of the problem, a number of randomly seeded hyperparameter sets as presented in [32]. The hyperparameter set with the lowest negative log-likelihood is chosen. Predictions are done as presented in Figure 3.7.

The SVM is trained using k-fold cross-validation on a grid of hyperparameters C and γ for the squared exponential kernel. The problem is usually solved in the primal-dual relationship [50, 38]. The formulation is for two classes, with labels $t_i \in \{-1, 1\}$. The original problem was

$$\underset{\mathbf{x},\xi,\mathbf{b}}{\text{argmin}} \frac{1}{2}||\mathbf{w}||^2 + C\sum_{i=1}^{n} \xi_i. \tag{3.1}$$

subject to $t_i(\mathbf{w} \cdot \phi(\mathbf{x}_i) + \mathbf{b}) \geq 1 - \xi_i$. Is modified

$$\underset{\alpha}{\operatorname{argmin}}\, \alpha^T Q \alpha - \mathbf{e}^T \alpha. \tag{3.2}$$

where $\mathbf{e}$ is a n long vector of ones, Q is an $n \times n$ positive semidefinite matrix with entries $Q_{ij} \equiv t_i t_j K(\mathbf{x}_i, \mathbf{x}_j)$. The problem is subject to $\mathbb{t}^T \alpha = 0$ and $0 \leq \alpha_i \leq C$ for all training configurations i. The problem is solved only during prediction, when t_i, α, b, C, all nn support vectors and kernel parameter γ are stored and later used during prediction. The prediction for an unevaluated configuration $\mathbf{x}_*$ is performed as follows using the previously stored data

$$d(\mathbf{x}_*) = \operatorname{sgn}(\sum_{1}^{nn} t_i \alpha_i k(\mathbf{x}_i, \mathbf{x}) + b). \tag{3.3}$$

where sgn is the sign function. When there are more than two classes present, "one-against-one" approach is used [112] to compute the decision function.

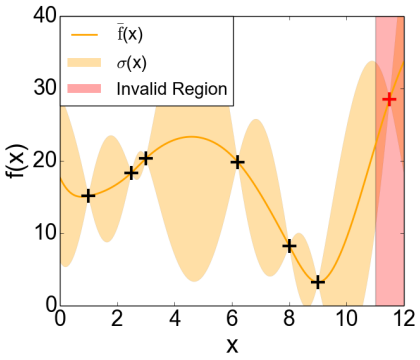


Figure 3.8: MLO surrogate model.

3.2.3 Particle Motion

The motion of particles is governed by the classical PSO equations, although implementing some alterations due to the discrete parameter space and integration of the surrogate model. The classical PSO equations governing particle motion are

$$r_1 \sim U(0,1), \qquad r_2 \sim U(0,1). \tag{3.4}$$

$$v_i = w v_i + c_1 r_1 (l_i - x_i) + c_2 r_2 (g_i - x_i). \tag{3.5}$$

$$x_i = x_i + v_i. \tag{3.6}$$

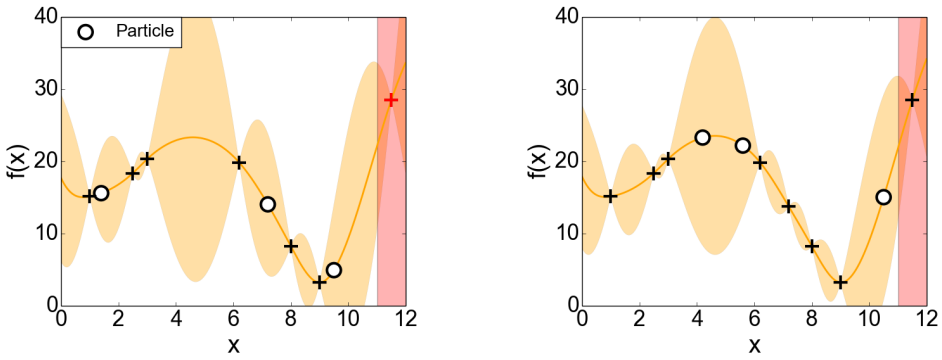
where i is dimension index, r_1 and r_2 are uniform random numbers, $\mathbf{l}_*$ is the particles $\mathbf{x}_*$ so far local best found position, $\mathbf{g}_*$ is the global so far best found position and $U(0,1)$ is a uniform random number with range $[0,1]$. The global best found position is shared across all of the particles. The v_i and are particle velocities, c_1 and c_2 are the acceleration coefficients and w is the inertia weight. The equations are defined for continuous $\mathbb{R}^D$ spaces. 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 any of the parameters is discrete. The PSO algorithm can be modified to suit discrete spaces [36, 123, 122], and can even be defined for binary spaces [120]. The approach is to either map and demap the problem to the continuous domain, where classical PSO can be applied or to redefine the particle motion governing equations. The MLO algorithm uses dithering to solve the problem, which is essentially the mapping approach. The resulting truncation is not visible if the space cardinality is high, making the space seem continuous with truncation having little impact. To discretize the position value of a particle after its movement, its value is rounded. The rounding error is randomized (dithering) as presented in Eq. 3.7. By using dithering instead of truncation PSO particles maintain their velocity component which results in a more thorough exploration.

$$dither(x_i) = \begin{cases} \lfloor x_i \rfloor & U(0,s_i) \geq (x_i \mod s_i\ ). \\ \lceil x_i \rceil & U(0,s_i) < (x_i \mod s_i\ ). \end{cases} \tag{3.7}$$

where s_i is the minimal increment for the respective parameter.

The model is integrated as follows. For all $\mathbf{x}_*$ predicted to lie in $\mathcal{V}$: Whenever $\sigma(\mathbf{x}_*)$ returned by the GP is below a credible interval min_σ the predicted mean $\bar{f}(\mathbf{x}_*)$ is used; otherwise the prediction is deemed to be inaccurate and design configuration evaluation follows $f(\mathbf{x}_*)$. The metaheuristic will avoid $\mathcal{I}$ region, as values predicted to lie outside of the valid region are assigned unfavorable - ∞ value. The mechanism is analogical to the one presented in [72], although based on surrogate models. It is illustrated in Figure 3.9 and encapsulated in the function fit, used to assign fitness to particles

$$fit(\mathbf{x}_*) = \begin{cases} \bar{f}(\mathbf{x}_*) & \sigma(\mathbf{x}_*) > min_\sigma \text{ and } d(\mathbf{x}_*) = 0. \\ f(\mathbf{x}_*) & \sigma(\mathbf{x}_*) \leq min_\sigma \text{ and } d(\mathbf{x}_*) = 0. \\ -\infty & d(\mathbf{x}_*) \neq 0. \end{cases} \tag{3.8}$$



(a) *Iteration N* (b) *Iteration N+1*

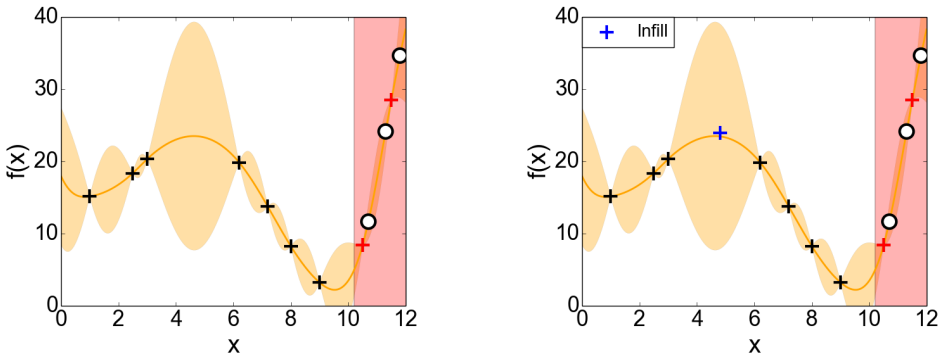
Figure 3.9: When calculating particle motion, depending on the uncertaintity prediction $\sigma(\mathbf{x}_*)$, a particle either evalautes a configuration $f(\mathbf{x}_*)$ or uses the models fitness prediction. In (a) the highlighted particle exceeded the min_σ, subsequently the configuration was evaluated. In (b) the particles move towards promising eareas.

The PSO algorithm proceeds for up to $\mathfrak{m}$ iterations without design configuration evaluation, which is explained in details in the next subsection. To allow particles to move from one edge of the parameter space to the other in $\mathfrak{m}$ iterations, while preventing excess velocity build up, a velocity clamping mechanism is employed [58]. Particles can traverse the whole parameter space in at most $\mathfrak{m}$ iterations, before the global best found design is evaluated.

$$v_i = \min(\frac{s_i}{\mathfrak{m}}, v_i). \tag{3.9}$$

3.2.4 Evaluate Best or Infill

The PSO algorithm proceeds for up to $\mathfrak{m}$ iterations without design configuration evaluation if the predicted standard deviation $\sigma(\mathbf{x}_*)$ for all $\mathbf{x}_*$'s is below a credible interval. Every $\mathfrak{m}$ iterations; the global best found design configuration $\mathbf{g}_*$ is evaluated as suggested in [66]. If it was already evaluated, an infill configuration or a small perturbation of the $\mathbf{g}_*$ is evaluated. If there are no unevaluated configurations available within the valid region a perturbed configuration is evaluated. The perturbation is described by a vector of D uniform random numbers $U(-1, 1)$ within a range of one step s in respective dimensions $(\mathbf{s} \circ U(-1, 1)^D)$, similar to the concept described in [66]. A small perturbation of the best configuration can allow for the refinement of the model and for the optimization to proceed.



(a) *Particles Stalled* (b) *Infill*

Figure 3.10: In (a) all of the particles their local best found so far positions $\mathbf{l}_*$ reside in the invalid region. The PSO equations break. The infill (b) evaluates a configuration with highest mean across the parameter space, and subsequently updates the global found so far position $\mathbf{g}_*$. The particle dynamics restarts.

$$f(dither(\mathbf{g}_* + \mathbf{s} \circ U(-1,1)^D)). \tag{3.10}$$

The perturbation is dithered similar to particle motion. The infill process finds an unevaluated configuration with maximum predicted fitness. The infill is performed over the valid region as predicted by the decision function constructed using the SVM classifier.

$$\underset{\mathbf{x}_* \in \mathcal{V}}{\operatorname{argmax}} \, \bar{f}(\mathbf{x}_*). \tag{3.11}$$

3.2.5 Termination

Although the MLO will converge towards an optimum, it is limited by heuristic search restrictions and, as such, it cannot be guaranteed to find the global optimum. Hence, it is crucial to specify termination criteria. There are two possible MLO termination criteria. First, MLO terminates if it exhausts its allocated compute time budget. For example, a number of machines is allocated for a 24-hour period. Alternatively, the MLO algorithm terminates when a parameter configuration $\mathbf{x}$ is found that achieves user required performance.

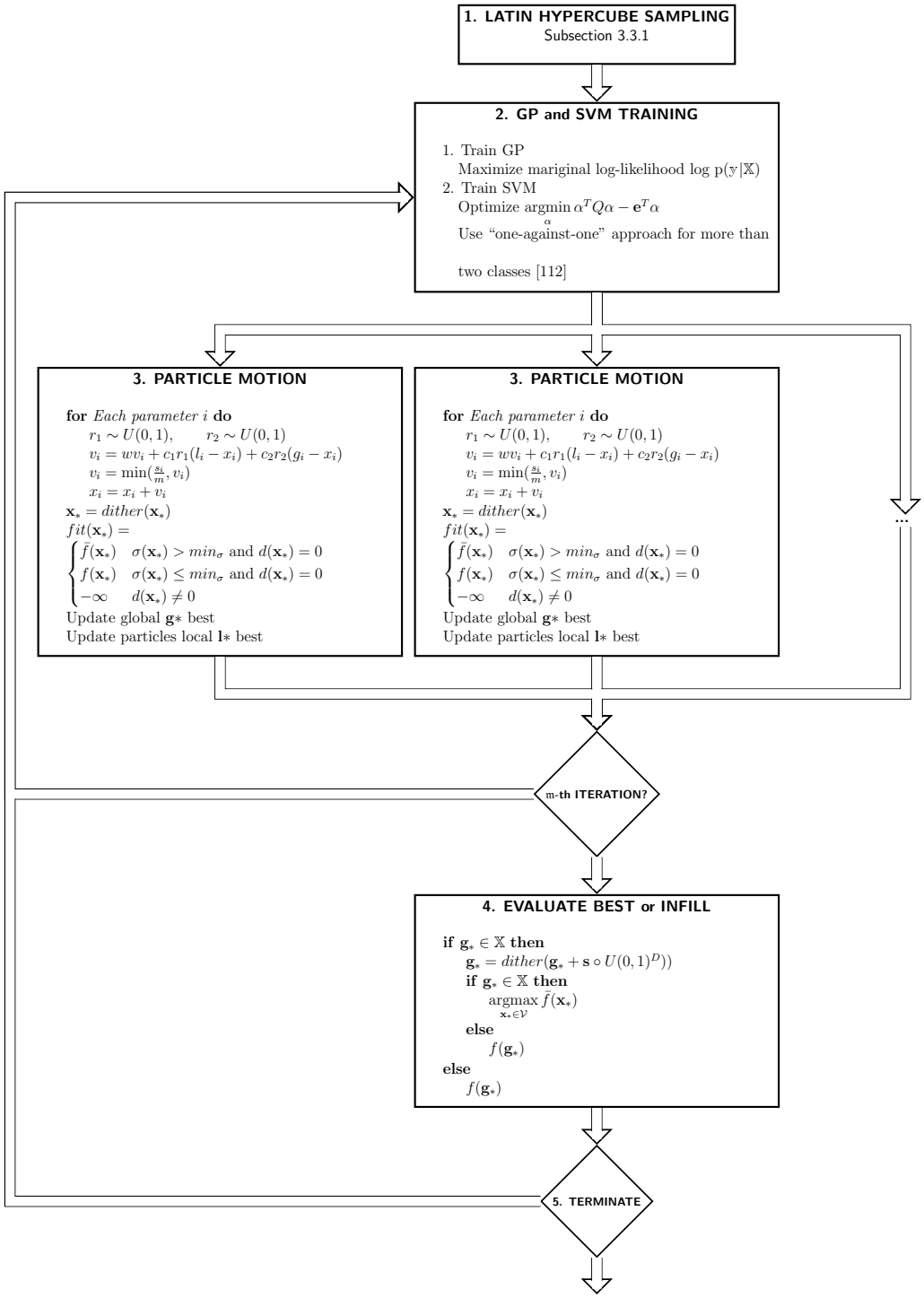


Figure 3.11: MLO Algorithm with detailed data flow.

3.3 Evaluation

The two evaluation goals are to assess MLO optimization performance sensitivity with respect to its parameters and the impact of infill on algorithm's performance. For evaluation, the MLO approach is used to optimize four designs and a comparison is made with a hill climbing algorithm. The portfolio of benchmark designs is meant to evaluate different aspects of the new algorithm. Although some performance models for the benchmark designs are available or can be constructed, they are not used in optimization. The evaluated generic algorithms treat the benchmark designs as defined in the problem statement Subsection 2.2.2, otherwise they would not be generic anymore. Each of the mentioned designs had hardware generated prior to the optimization to make the experiments repeatable. The task involved generation and analysis of hundreds of bitstreams over a time period of a couple of months.

The first design is a quadrature-based financial design with variable precision [155]. Two benchmarks are provided with the design, one which evaluates energy efficiency and one which evaluates throughput of the design. The design parameters involve numerical representation, parallelism and degree of recomputation. The design is constrained by accuracy of the output dependent on the numerical representation. The numerical representation and parallelism trade off accuracy and performance. It is a commonly found pattern in reconfigurable computing.

The second design is a PQ design with variable precision, number of computational cores and design clock frequency [46]. The design is constrained both by timing and the number of resources. Higher clock frequency allows for better performance, yet make hardware generation more difficult. Increasing resource utilization improves performance of the design by utilizing more cores and decreasing amount of necessary recomputation at the expense of making PAR more difficult. Timing and PAR difficulties are found in nearly all reconfigurable designs.

The third and fourth designs are the robot localization design and stochastic volatility financial designs, both based on the sequential Monte Carlo SMCGen framework [47]. In both cases one of the parameters has no impact on the performance of the design. This is an important feature which allows for evaluation of the algorithms ability to detect irrelevant parameters. Also, the performance of the design measured by the benchmark is very noisy making the optimization potentially difficult. The two design fitness functions have different degrees of additive noise, and their fitness functions exhibit different behavior.

A number of different configurations of MLO are evaluated. MLO is evaluated

Table 3.1: MLO test designs overview.

Design	**Noise**	**Constraints**	**No.**[II]	*D*
Quadrature$^{\delta}$ [155]	Performance of the design exhibits some noise. One of the parameters is a software parameter. Two benchmarks are available, a design throughput and an energy efficiency benchmarks. LUT bound	Accuracy and resource constraints.	20,000	3
PQ$^{\delta}$ [46]	Some of the designs fail randomly due to timing and PAR difficulties. LUT bound.	Resource constraints$^{\omega}$.	8,200	3
Robot$^{\delta}$ [47]	Noisy benchmark function. The challenge for efficient optimization is for the algorithm to determine that one of the parameters has no impact on the design's performance.	Resource and Accuracy constraints. One of the parameters is a software parameter.	24,000	3
Stochastic$^{\delta}$ [47]	Similar to the previous design. Noisy benchmark function. The challenge for efficient optimization is for the algorithm to determine that one of the parameters has no impact on the design's performance.	Resource and Accuracy constraints. One of the parameters is a software parameter.	24,000	3

II Number of possible designs in the parameter space.

δ Optimized for Maxeler MPC-X1000 system with a Xilinx Virtex-6 XC6VSX475T FPGA.

ω The biggest challenge with optimization of the PQ design is PAR and timing issues.

using three different GP kernel functions, each with additive Gaussian noise. SVM is chosen as the classifier with an squared exponential kernel which is cross-validates on a set of parameters $\gamma \times C = \{1.2^i\}_{i=-10}^{10} \times 10\{1.25^i\}_{i=1}^{10}$. There are many possible grid settings which could offer similar performance, this particular choice is based on previous experience. The PSO constants c_1 and c_2 are set to 2.0 as recommended [58]. The following parameters are set to $\mathfrak{m} = 10$ and $min_\sigma = 0.01$, unless otherwise noted. To evaluate the MLO performance optimization terminates after 200 configuration evaluations or when the global optimum is found. For evaluation purposes, the global optimum for each design is determined using exhaustive search prior to experiments.

3.3.1 Implementation

The MLO algorithm is implemented in Python using DEAP (Distributed Evolutionary Algorithms in Python library) [52], PyGPs (Python Relational Gaussian Process Regression library) [19] and Scikit-learn (Python Machine Learning toolkit) [115]. The PyGPs framework is used to implement GP regression and Scikit-learn is used to implement SVM classification. To allow for rapid optimization, flexible python wrappers suitable for any API are provided. The MLO implementation calculates the fitness function by generating a bitstream using the supplied parameters and running a test benchmark. The benchmark can be used to examine design in terms of power usage, area, throughput or other metrics. Desings were synthesized using MaxCompiler for the Maxeler MPC-X1000 system with Xilinx Virtex-6 XC6VSX475T FPGA.

The algorithm terminates hardware generation if, during the preliminary resource report, any of the resources exceeds the FPGA size by more than 10%. This is crucial for automated optimization, the preliminary reports give a good indication of the final resource usage and likely overmapping can be detected as quickly as in 20 minutes. Compared to full hardware generation, a single quadrature design on Xilinx Virtex-6 XC6VSX475T FPGA takes up to 5 hours and of PQ design of up to 14 hours. In the worst case, for the robot localization design, it took as much as 30 hours for a single hardware generation.

3.3.2 Quadrature-based Financial Design

In [155] the designer explores the trade-off between accuracy and throughput in a quadrature-based financial design with three parameters. The design can be used to compute integrals for various financial applications. The first two parameters are mantissa width m_w of the floating point operators and the number of computational cores κ. Larger

number of m_w bits increases computation accuracy, but limits the maximum number of κ 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. All of the parameters are uniformly discrete.

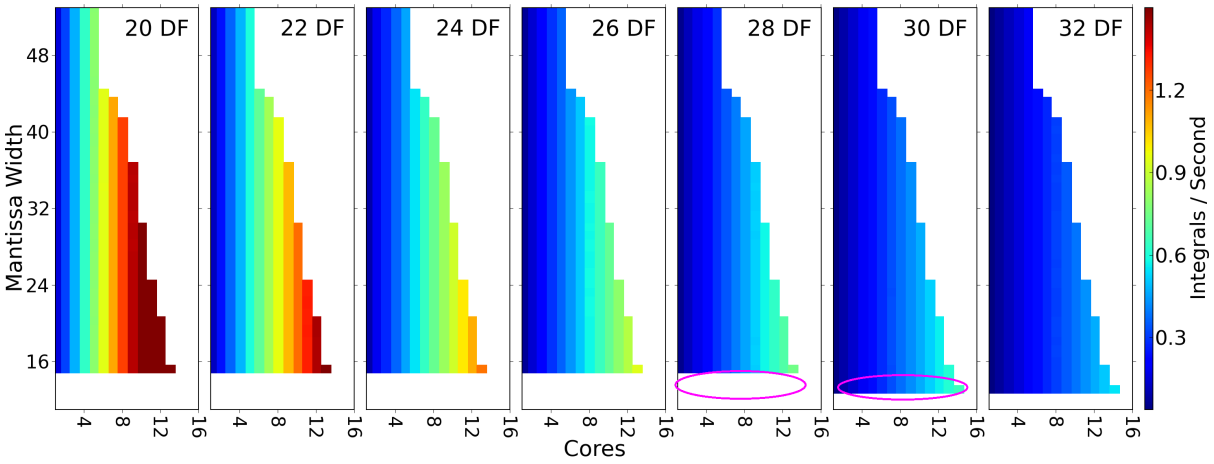


Figure 3.12: Visualization of a subset of the parameter space for the throughput benchmark of the Quadrature-based Financial design when $\epsilon_{rms} = 0.1$. Area affected by decreasing d_f is highlighted.

The optimization goal is to find the design offering the highest throughput of integrations per second ϕ_{int} or lowest energy cost per integral throughput W/ϕ_{int} 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. Although some designs produce inaccurate results, the results can be reused for regression. Resource usage is linearly related to κ. Density factor d_f is a software parameter while m_w and κ affect the bitstream. Varying d_f only involves software execution, as long as a bitstream for the given m_w was already generated. If a design with m_w, κ is evaluated that has not been evaluated before, a new bitstream is generated. The design with highest throughput is not always equivalent to the most energy efficient design.

Throughput Optimization

The throughput of the quadrature-method based design is analysed. Throughput optimization is a maximization problem. The true fitness function of quadrature design energy efficiency for $\epsilon_{rms} = 0.1$ is presented in Figure 3.12.

MLO clearly outperforms the hill climbing algorithm, and offers better performance than the analytical approach. Optimization visualizations for different ϵ_{rms} using different infill methods are shown in Figures 3.13-3.15. It is visible that MLO with infill on mean outperforms other configurations, especially prominent when $\epsilon_{rms} = 0.1$ in Figure 3.13. All configuration outperform the analytical approach, which always takes 198 hours to find the optimal design. The reduction in optimization time is of up to 50% when $\epsilon_{rms} = 0.01$. The optimization time for the hill climbing algorithm follows a reverse trend with regards to ϵ_{rms} and the optimization time than MLO. This is mainly thanks to SVM classifier parameter space trimming. Hill climbing algorithm always starts from the same starting point, and with more restrictive ϵ_{rms} limit the path to reach the global optimal design is longer. In the MLO case, the parameter space trimmed when ϵ_{rms} is decreased hence optimization is faster.

Optimization visualizations for different ϵ_{rms} using different kernel functions are shown in Figures 3.16-3.18. The squared exponential kernel with ARD offers the best performance, aside of the case when $\epsilon_{rms} = 0.01$. For $\epsilon_{rms} = 0.1$ seen in Figure 3.16 the Matèrn kernel, marked in green offers, a noticeably lower performance. It is important to note, that the differences although noticeable, are not big enough to rule out a clearly better performing kernel.

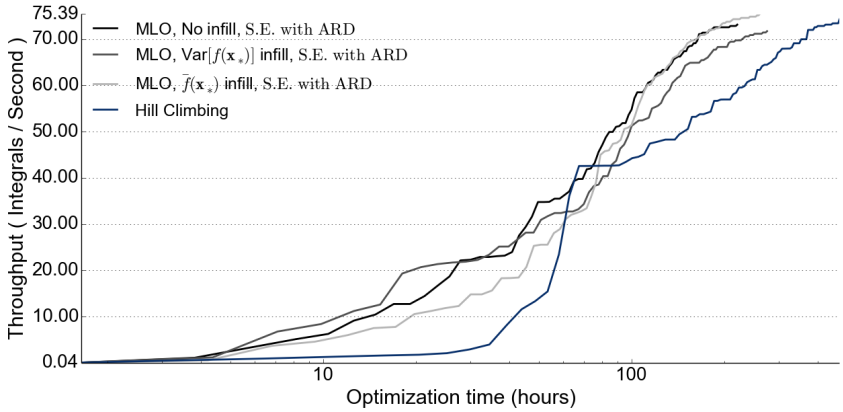


Figure 3.13: Optimization of the Quadrature-based Financial design throughput benchmark $\epsilon_{rms} = 0.1$ using different infill functions.

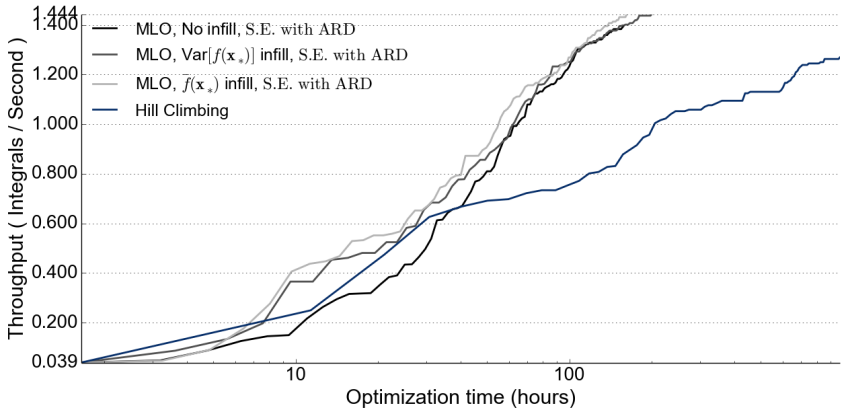


Figure 3.14: Optimization of the Quadrature-based Financial design throughput benchmark $\epsilon_{rms} = 0.01$ using different infill functions.

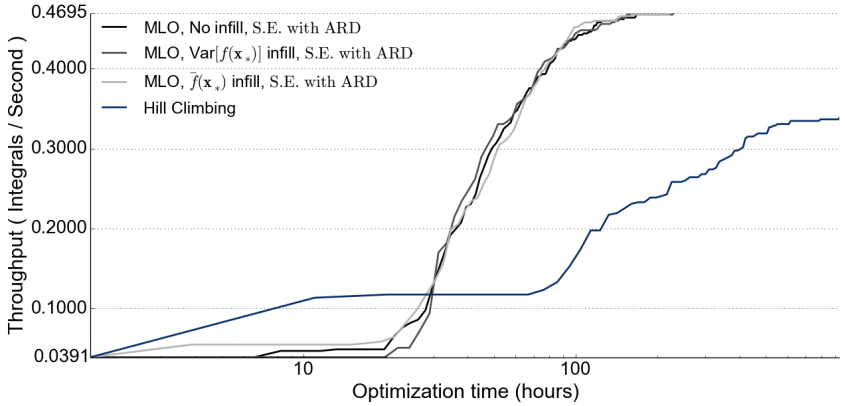


Figure 3.15: Optimization of the Quadrature-based Financial design throughput benchmark $\epsilon_{rms} = 0.001$ using different infill functions.

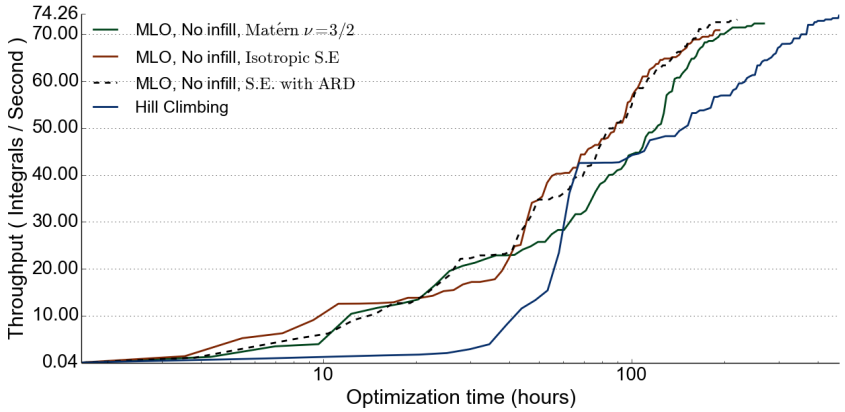


Figure 3.16: Optimization of the Quadrature-based Financial design throughput benchmark $\epsilon_{rms} = 0.1$ using different kernel functions.

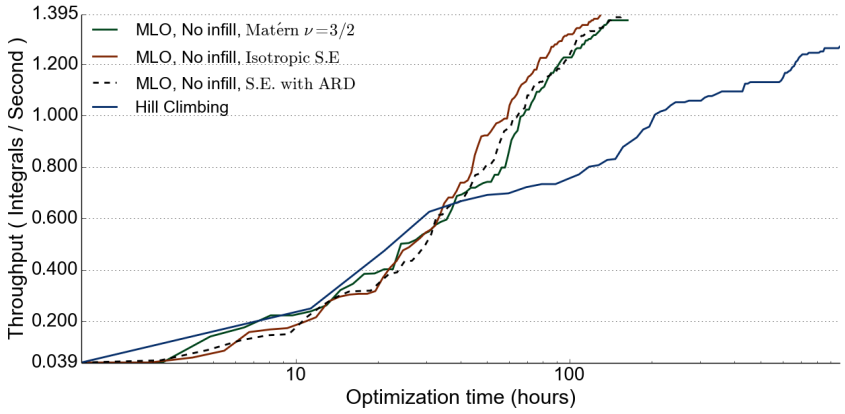


Figure 3.17: Optimization of the Quadrature-based Financial design throughput benchmark $\epsilon_{rms} = 0.01$ using different kernel functions.

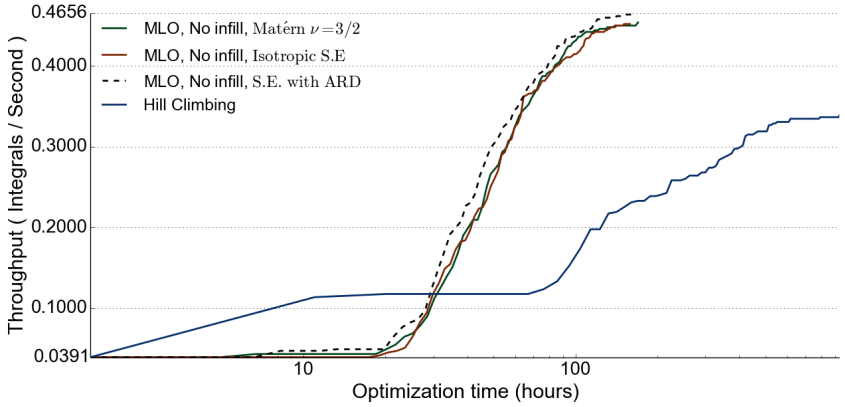


Figure 3.18: Optimization of the Quadrature-based Financial design throughput benchmark $\epsilon_{rms} = 0.001$ using different kernel functions.

Energy Efficiency

The energy consumption per computation is analysed and poses a minimization problem. Energy consumption is measured using Maxeler Technologies provided tools. The biggest difference between throughput and energy efficiency optimization is the rate at which f varies; the function changes more rapidly in the throughput case. The optimal configuration is not always throughput optimal. The valid regions for all ϵ_{rms} limits are identical to the throughput benchmark.

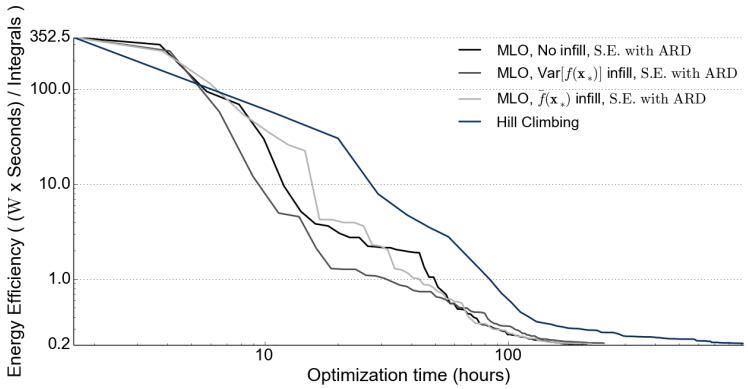


Figure 3.19: Optimization of the Quadrature-based Financial design energy benchmark $\epsilon_{rms} = 0.1$ using different infill functions.

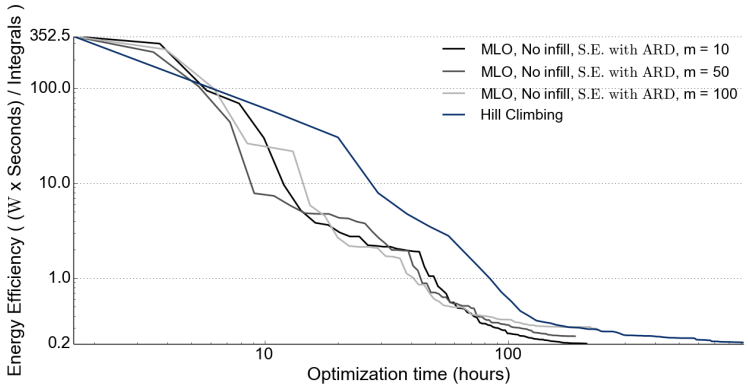


Figure 3.20: Optimization of the Quadrature-based Financial design energy benchmark $\epsilon_{rms} = 0.1$ using different $\mathfrak{m}$ values.

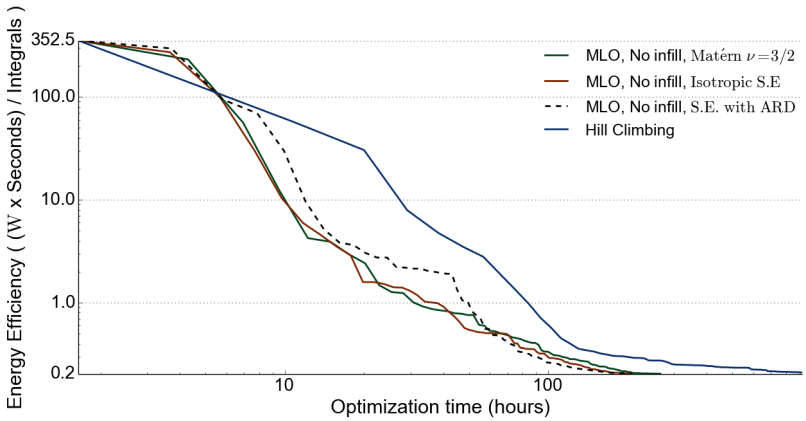


Figure 3.21: Optimization of the Quadrature-based Financial design throughput benchmark $\epsilon_{rms} = 0.1$ using different kernel functions.

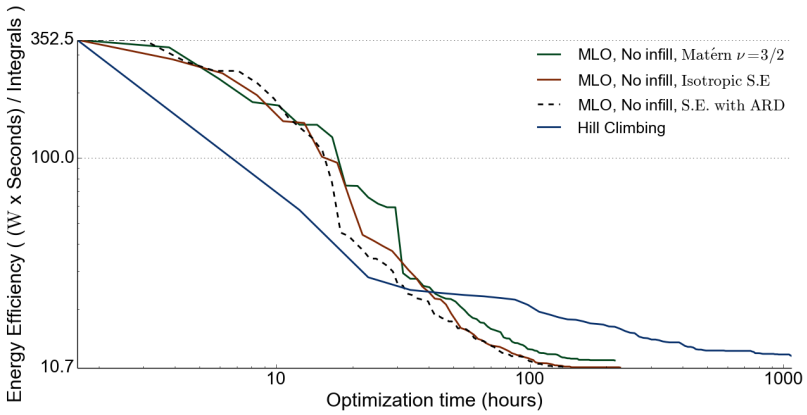


Figure 3.22: Optimization of the Quadrature-based Financial design throughput benchmark $\epsilon_{rms} = 0.01$ using different kernel functions.

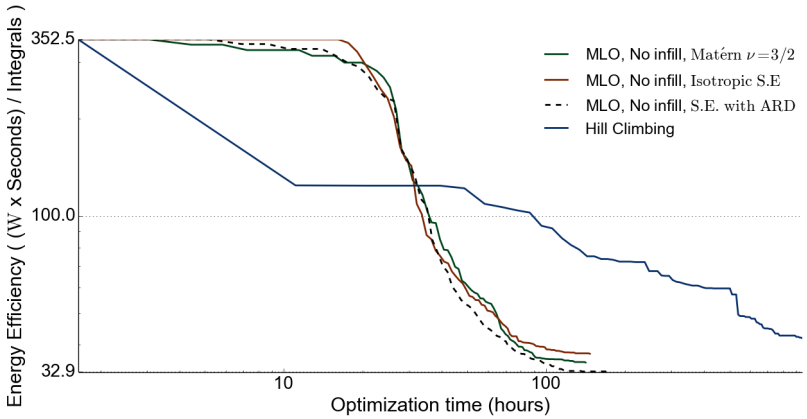


Figure 3.23: Optimization of the Quadrature-based Financial design throughput benchmark $\epsilon_{rms} = 0.001$ using different kernel functions.

The infill has no obvious impact on the optimization, as seen in Figure 3.19. In Figure 3.20 impact of the $\mathfrak{m}$ parameter is shown, where increasing $\mathfrak{m}$ clearly decreases algorithm performance. Visible in Figures 3.21-3.23 the squared exponential kernel with ARD offers the best performance, although again the benefit is not drastic. As in the throughput optimization case, the trend for the hill climbing and MLO is reversed with respect to the optimization time and ϵ_{rms} limit.

3.3.3 Real-time Proximity Query (PQ) Design

In [46] the authors study a real-time PQ design. PQ involves computing the intersection or the closest point-pair between two objects in 3D. It is particularly useful in robot

motion planning, haptics rendering, virtual prototyping, computer graphics, animation, and imaged-guided surgical robotics. The authors adopt a reduced precision data format to reduce logic utilization and perform re-computation using high precision when necessary to preserve the accuracy of results.

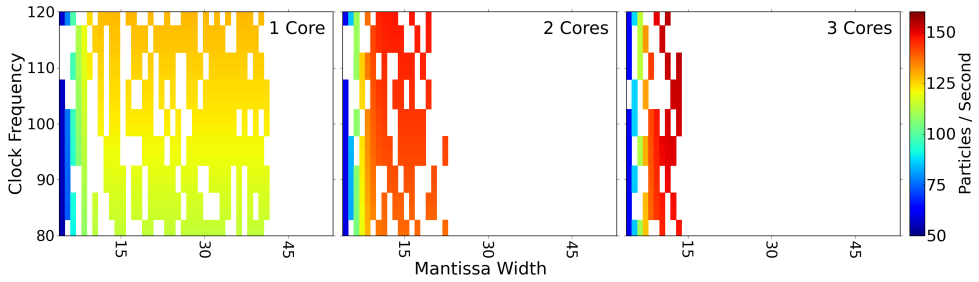


Figure 3.24: PQ design throughput fitness function visualization [46]. There is a clear trend where with increased number of cores smaller number of designs become available due to resource constraints. Random patches represented by blank of invalid area are the result of timing and PAR issues.

The optimization goal is to find the design offering the highest throughput of points per second under the effect of three parameters. The first two parameters are the mantissa width m_w of the floating point operators and the number of computational cores κ. A smaller number of m_w bits increases the number of cores that can be implemented on the chip because the size of each core is reduced, but the accuracy decreases and more re-computations are necessary. There is a chip resource limitation so one cannot increase both m_w and κ for the lowest ratio of re-computation and the highest level of parallelism. The third parameter is the clock frequency $freq$ of the computational cores. Increasing $freq$ reduces the value of κ because the placement and routing process needs more space to meet the timing requirement. The frequency can be set between 80 MHz and 120 MHz, there are between 1 and 4 cores and m_w can be anything between 4 and 53. This results in a total of around 8,000 possible design configurations.

The PQ design is highly noisy, as seen in Figure 3.24. Multiple designs fail when unexpected and vice-versa due to timing constraints; only one cost table was used per design. For a given setting a four core design might run at 120 MHz, but not at 115 MHz. Meanwhile three core design could succeed for 115 MHz but not 120 MHz. This is challenging for the SVM classifier to correctly predict valid designs to allow for optimization, while pruning parameter space regions which contain designs likely to fail. The PQ design is challenging due to frequency tuning, timing issues cannot be as accurately predicted as resource overmaping. The model presented by the authors of the PQ design [46] could be exploited in order to use a design automated approach, yet there

are several difficulties. The difficulty in generating hardware running at certain frequency is dependent on the configuration of the design, and is impractical to model. Hence, no comparison against the design specific approach is provided.

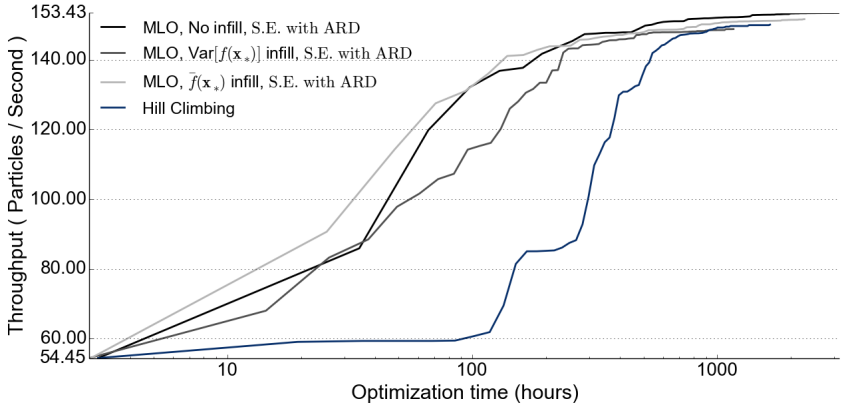


Figure 3.25: Optimization of the PQ design using different infill functions. MLO configuration is the anistropic squared exponential kernel function and $min_\sigma = 0.01$.

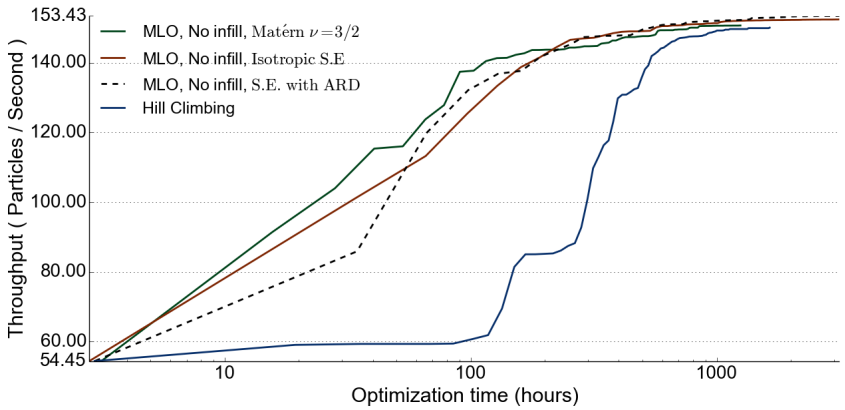
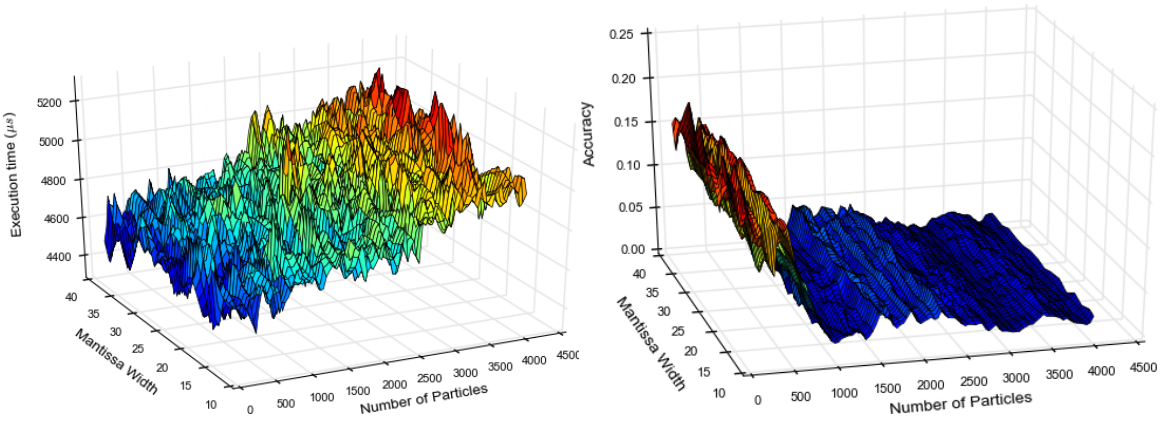


Figure 3.26: Optimization of the PQ design using different kernel functions. MLO configuration is no infill and $min_\sigma = 0.01$.

The optimization of the PQ design using MLO with different infill functions is presented in Figure 3.25. The hill climbing algorithm initially struggles to navigate through the noisy space, hence its low performance. Although MLO and hill climbing algorithms offer similar design performance around 1000 hours of optimization time, the average case is much better for the MLO algorithm. This can be clearly seen in Figure 4.14. MLO with no infill offers marginally better performance than infill on mean. Infill on variance offers noticeably worse performance around the 100 hour mark.

Comparison of MLO with different kernels is presented in Figure 3.26. Both the isotropic and squared exponential kernel with ARD offer similar performance. The Matèrn kernel offers better performance during earlier stages of optimization, up till around 100 hours, being superseded by other kernels later on. The $\mathfrak{m}$ parameter does not have a noticeable impact on the performance of the algorithm.



(a) *Stochastic volatility f* (b) *Stochastic volatility accuracy*

Figure 3.27: Visualization of the stochastic volatiltiy design with a single core. The left image shows the execution time, while the right image shows the accuracy.

3.3.4 Stochastic Volatility Design

The design is implemented using the SMCGen framework [47]. The goal of SMC methods is to estimate the posterior distribution of some hidden problem states. In particular, SMC deals with problems where new observations come in a sequence and inference has to be done on-line. SMC are simulation based methods, where a number of particles is used to model the posterior distribution. Stochastic volatility models are often used in finance when there is no closed form solution to a problem, or the user does not wish to use approximations to pricing functions due to accuracy constraints. The volatility is then modeled as a stochastic process. SMC methods are one possible solution when the inference has to be done online. For this particular design [47], the parameters are the number of processing cores N_C, the number of particles used in the simulation N_P and mantissa width of numerical operators m_w. The number of cores N_C is limited to 64 in powers of two, the number of particles is tested on the range of 96 to 3984 in 96 increments and the range of the mantissa width m_w of the floating point operators is set from 10 to 40. The number of particles N_P is a software parameter. All of the parameters are uniformly discrete.

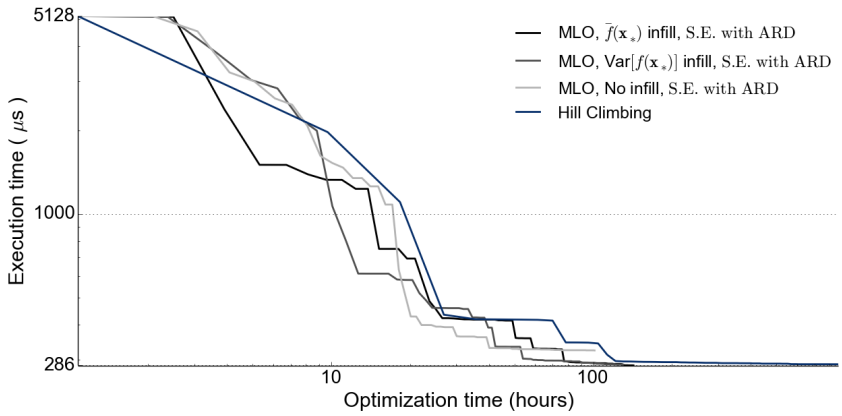


Figure 3.28: Optimization of the stochastic volatility execution time benchmark using different infill functions.

The software benchmark was executed 300 times, the execution time of it is in the microsecond range and therefore is highly susceptible to measurement noises. The process took as much as 5-6 minutes per parameter setting. The noise is clearly visible in Figure 3.27. Both the accuracy and the fitness function are noisy. The challenge in optimization of the stochastic design is for the algorithm to discover that m_w has no impact on both the accuracy and the execution time. The chip cannot accommodate more than 16 cores. The design suffers from a minor place and route as well as timing issues.

In Figure 3.28 the optimization using MLO and different infill functions is presented. The infill function has minor impact on the performance. Although MLO offers better performance than the hill climbing algorithm, the difference is not as noticeable as in the previous cases. It is an ideal scenario for the hill climbing algorithm, it can take a path where it rarely generates hardware. Furthermore, as seen in Figure 3.27(b) the m_w has neither any impact on the accuracy or the performance of the design. This means that the hill climbing algorithm has a relatively straightforward path to the optimal design.

The impact of kernel functions on performance of MLO is presented in Figure 3.29. For all of the algorithms there is a noticeable improvement in design's performance between the 10th and 30th hours. Yet, it is far more prominent for MLO with isotropic squared exponential kernel with an odd drastic improvement around the 20th hour. It is possibly because the isotropic kernel can quickly and accurately model the problem requiring few parameter setting evaluations. The alternative is a not sufficiently high number of experiments. There is a similar, although not as drastic, improvement for the squared exponential kernel with ARD.

The curves presented in both Figure 3.28 and Figure 3.28 are more edgy than in the

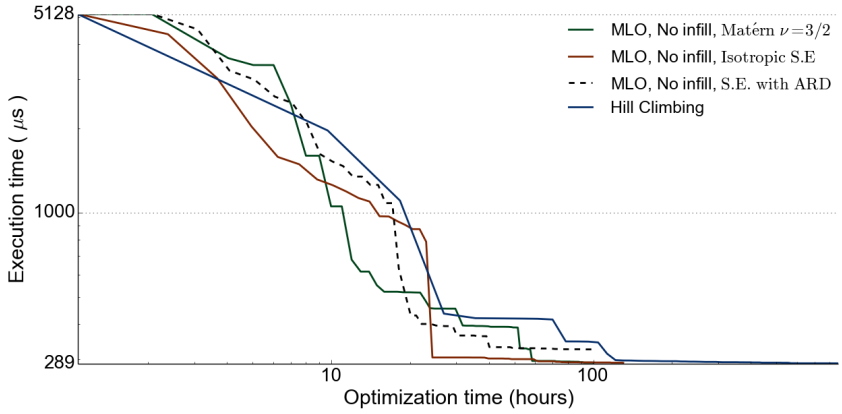


Figure 3.29: Optimization of the stochastic volatility execution time benchmark using different kernel functions.

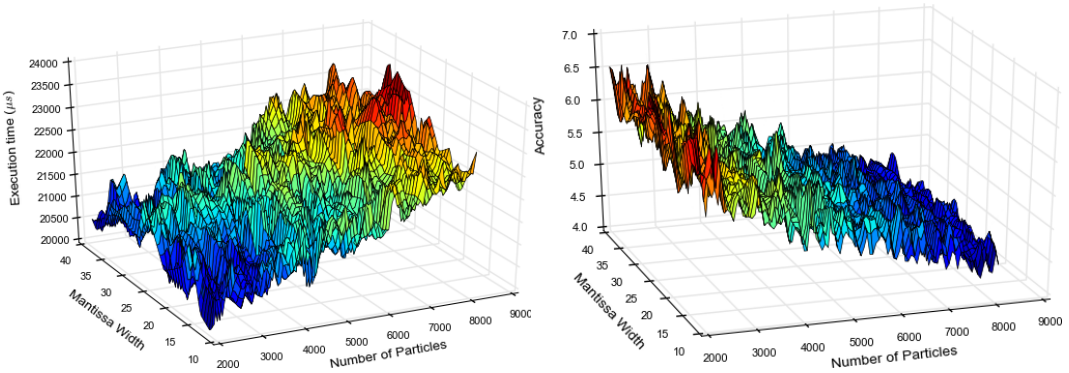
previous designs. This is because the frequency of design improvement is much lower, hence the curves are averages of a smaller number of points.

3.3.5 Robot Localization

The robot localization design is implemented using the same SMCGen framework [47] as the stochastic volatility design. The design is used for mobile robot localization. The robot needs to be aware of surrounding moving objects. In a sequential loop fashion the robot uses its sensors to identify its location and perform motion. The parameters are the number of processing cores N_C, the number of particles used in the simulation N_P and mantissa width of numerical operators m_w. For this particular design [47], the parameters are the number of processing cores N_C, the number of particles used in the simulation N_P and mantissa width of numerical operators m_w. The number of cores N_C is limited to 64 in powers of two, the number of particles is tested on the range of 2048 to 8096 in 96 increments and the range of the mantissa width m_w of the floating point operators is set from 10 to 40. The number of particles N_P is a software parameter. All of the parameters are uniformly discrete.

As in the stochastic volatility design, the software benchmark was executed 300 times, the execution time of it is in the microsecond range and therefore is highly susceptible to measurement noises. The process took as much as 5-6 minutes per parameter setting. The noise is clearly visible in Figure 3.30. Both the accuracy and the fitness function are noisy. The accuracy function is linear for this design, clearly seen when comparing Figures 3.27(b) and Figures 3.30(b). Furthermore, the optimal configuration for the robot design is when configured with 4 not 16 cores. For larger m_w, when maximizing the

number of cores, the design suffers from place and route as well as timing problems. For $N_C = 4$, the hardware is not generated for nearly all of the parameter settings with m_w larger than 22. The challenge in optimization of the robot localization design is, again, for the algorithm to discover that m_w has no impact on both the accuracy and the execution time.



(a) *Robot localization f* (b) *Robot localization accuracy*

Figure 3.30: Visualization of the robot localization design for a single core. The left image shows the execution time, while the right image shows the accuracy.

The impact of the infill function is presented in Figure 3.31. The squared exponential kernel with ARD clearly offers worst performance. The hill climbing offers better performance than all of the configurations. When using variance infill function, MLO offers better performance than the hill climbing algorithm in the later stages of optimization. The kernel functions seem to follow similar trend as seen in the stochastic volatility design. In Figure 3.31 the squared exponential kernel with ARD offers worst performance, while the isotropic squared exponential kernel offers the best performance. The curves are again bumpy due to relatively low frequency of finding a better design. Figure 3.33 and Figure 3.34 present interaction of the infill and kernel functions. Variance infill function seems to offer superior performance. Mean infill function offers the worst performance when combined with the squared exponential with ARD kernel. At the same time squared exponential kernel function with ARD offers the best performance when combined with the variance infill function. The non-obvious interaction of MLO components is quite noticeable.

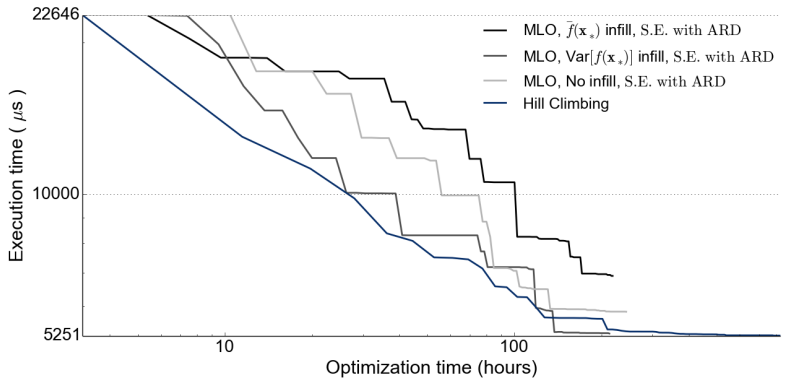


Figure 3.31: Optimization of the robot localization execution time benchmark using different infill functions.

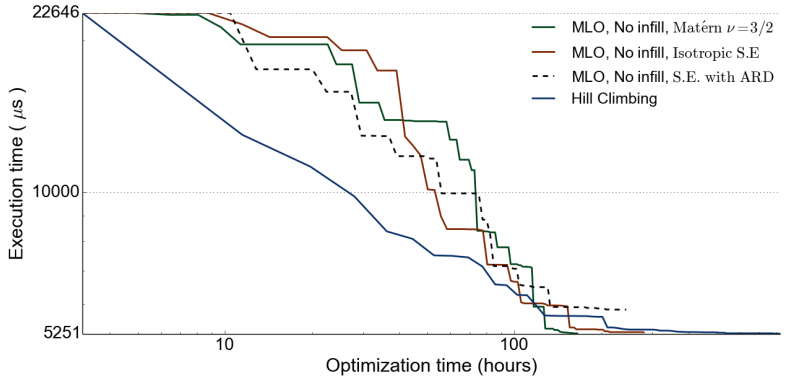


Figure 3.32: Optimization of the robot localization execution time benchmark using different kernel functions.

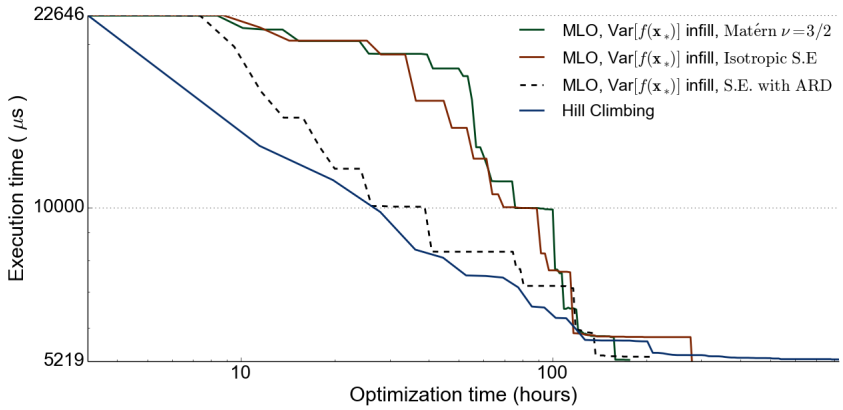


Figure 3.33: Optimization of the robot localization execution time benchmark using different kernel functions and variance infill function.

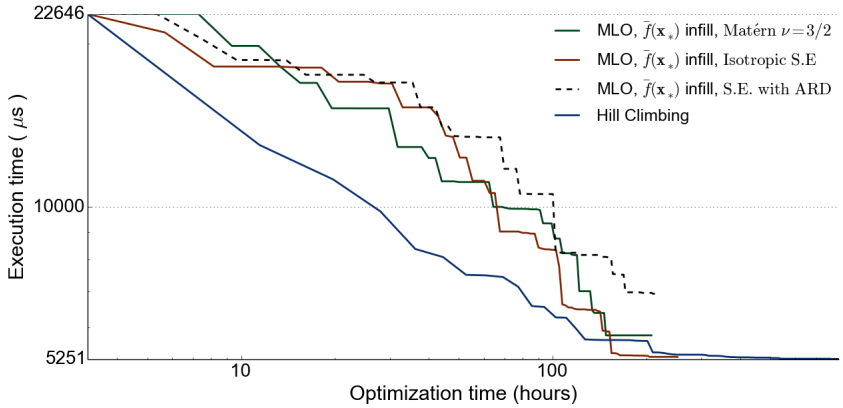


Figure 3.34: Optimization of the robot localization execution time benchmark using different kernel functions and mean infill function.

3.4 Discussion

3.4.1 Results

The MLO algorithm has two unique features. First, to the best of our knowledge, classifiers have not been integrated in surrogate model based algorithms for reconfigurable design optimization to decrease the search space size of reconfigurable designs. The advantage of this concept is a decreased parameter space, which should allow for faster convergence of the algorithm. This is essential in the case of reconfigurable designs, as it limits the number of lengthy hardware generations. Results of optimization with and without a classifier when using MLO algorithm are presented for Quadrature-based Financial design throughput benchmark in Figures 3.35-3.37. The classifier allows the algorithm to find the globally optimal configuration, as well as speeds-up termination of the algorithm. The same results where observed for the energy benchmark for the Quadrature-based Financial design. The benefit of using a classifier is clearly visible for the PQ design as seen in Figure 3.38. The benefit is less obvious than for the Quadrature-based design as larger portion of the optimization time is spent during the sampling process during which the classifier is not used. The benefit of using a classifier is most obvious when the design space is most limited, like for the Quadrature-based Financial design when $\epsilon_{rms} = 0.001$.

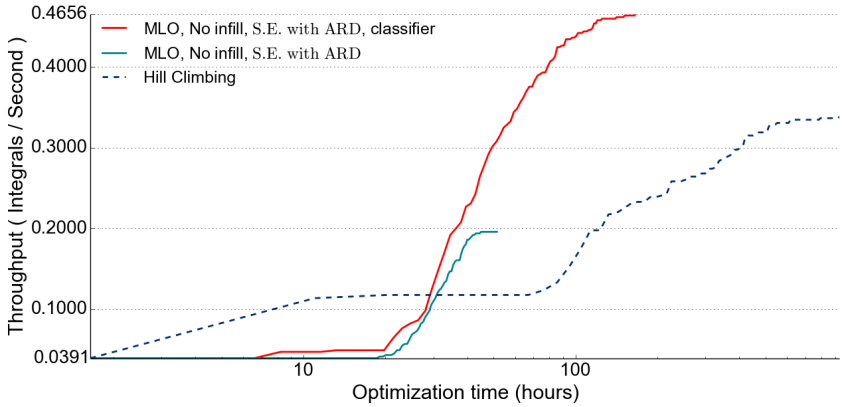


Figure 3.35: Optimization of the Quadrature-based Financial design throughput benchmark $\epsilon_{rms} = 0.001$ with and without a classifier.

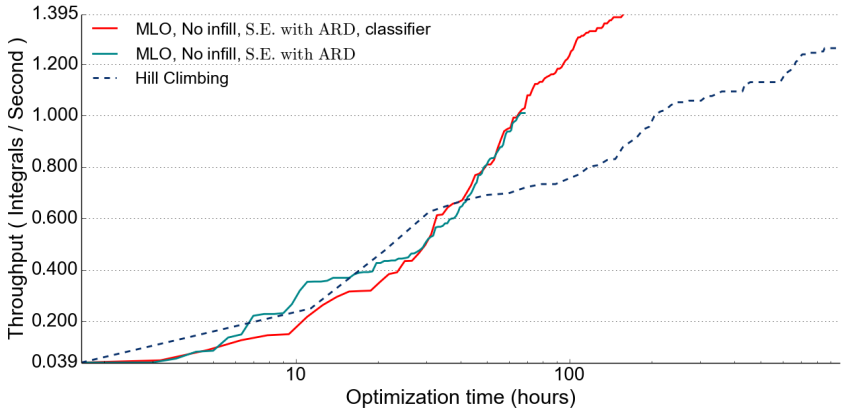


Figure 3.36: Optimization of the Quadrature-based Financial design throughput benchmark $\epsilon_{rms} = 0.01$ with and without a classifier.

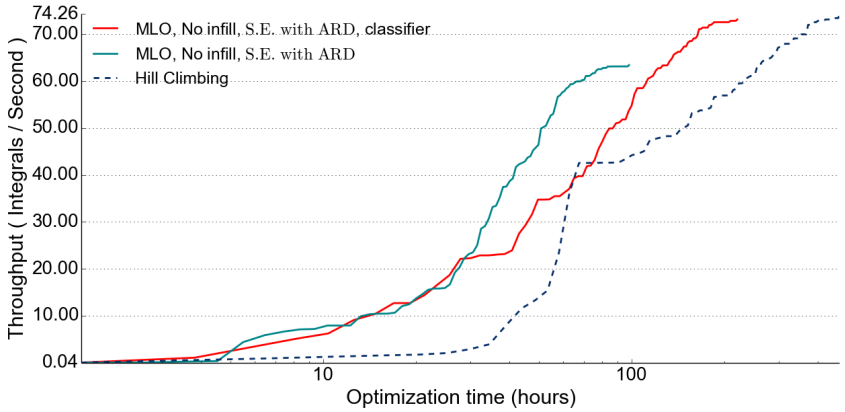


Figure 3.37: Optimization of the Quadrature-based Financial design throughput benchmark $\epsilon_{rms} = 0.1$ with and without a classifier.

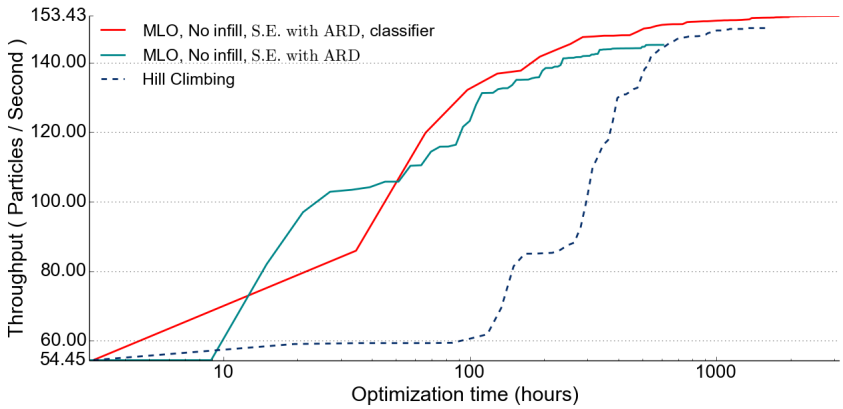


Figure 3.38: Optimization of the PQ design with and without a classifier.

The second and the new unique feature is the integration of an infill scheme to aid PSO. The construction of the surrogate model can fail for various reasons [78, 163, 146]. If the GP model fails to build, the standard deviation is too high for the metaheuristic to proceed, or particle swarm ceases dynamics infill parameter setting is evaluated. Three different infill strategies were evaluated. The results are summarized in Table 3.2. For the quadrature design, MLO finishes optimization in as little as 50% of the time taken by the design specific optimization tool, 99 hours vs. 198 hours. Generally, it offers between 3 to 5 times faster optimization than the hill climbing algorithm while offering better performing design configurations.

The mean infill function offers the best performance for both the throughput and energy benchmarks for the quadrature financial design. In nearly all cases the MLO algorithm finds the optimal design. The infill function does not offer good performance for the robot localization design. This is likely due to the fact that it is overly greedy, a strategy which is not always beneficial. The variance infill function only offered best performance for the stochastic volatility design. Generally, the trend is not clear. We opt for the mean infill function based on the reasoning that it offers better performance than variance infill function and it has one major advantage over not using infill. In case of designs suffering from place and route or timing problems, similar to the proximity query design, it is possible for region next to the global best parameter setting to consist solely of invalid parameter settings. In such case, evaluation of a perturbation is not going to result in hardware generation and improvement of the model. The algorithm effectively becomes a hill climbing algorithm.

Other parameters of the algorithm have been investigated, the $\mathfrak{m}$ parameter and kernel functions in particular. Increasing the $\mathfrak{m}$ parameter has a noticeably large impact on

Table 3.2: The average optimization time in hours and the percentage of the average performance of the optimal configuration of different designs. The hill climbing algorithm and three different configuration of the MLO algorithm are compared. The global optimum was found using exhaustive search.

Design	**H.C.**	**MLO** $\bar{f}(\mathbf{x}_*)$	**MLO** $\mathrm{Var}[\bar{f}(\mathbf{x}_*)]$	**MLO** no infill
Quad.Th. 0.1	300 (98%)	182 (**100%**)†	**190 (95%)**$^{-}$	151 (97%)
Quad.Th. 0.01	459 (88%)	125 (**100%**)†	137 (100%)	**115 (96%)**$^{\text{II}}$
Quad.Th. 0.001	393 (72%)	107 (**100%**)†	114 (100%)	104 (99%)
Quad.En. 0.1	562 (96%)	142 (97%)	**176 (96%)**$^{-}$	146 (99%)
Quad.En. 0.01	483 (88%)	102 (**100%**)†	133 (100%)	106 (99%)
Quad.En. 0.001	535 (78%)	102 (**100%**)†	105 (100%)	99 (100%)
PQ	903 (93%)	300 (91%)	300 (90%)	300 (92%)
Stochastic	337 (99%)	96 (99%)	84 (99%)	**54 (88%)**$^{\text{II}}$
Robot	366 (99%)	**153 (75%)**$^{-}$	162 (89%)	153 (99%)

1 The time presented is the average time when an algorithm stopped improving design's performance.

2 Numbers in parenthesis are the percentages of the performance of the optimal design configuration.

3 MLO use s.e. kernel function with ARD, $\mathfrak{m} = 10$ and $min_\sigma = 0.01$.

II Optimization time is short as the algorithm fails to find optimal design frequently.

− Worst performance out of all configurations.

† Infill on best mean finds the optimal configuration most frequently.

the algorithm in few cases, it is most prominent for the quadrature-based design. For the energy efficiency benchmark, when $\epsilon_{rms} = 0.1$, for MLO configured with $\mathfrak{m} = 100$ fails to improve the design's performance beyond 70-80% of the optimal configuration. The performance of the algorithm has a generally negative tendency when increasing $\mathfrak{m}$, although this is not always as noticeable.

When creating a surrogate model, the choice of the kernel function should be based on the domain specific knowledge of the problem. Most reconfigurable designs parameters spaces are spanned by different subspaces. The kernel function should ideally use ARD, being able to determine the relevance of different parameters on the design quality. For example the impact of numerical precision and the level of parallelism is different across the parameter space. At the same time, kernel functions with ARD have more degrees of freedom and require more data for accurate modeling. One would expect MLO with the squared exponential kernel function with ARD to perform best for both the stochastic volatility and the robot designs. This is not the case, as both of the designs have noisy fitness functions and they are largely linear. This means that the Matérn kernel and isotropic squared exponential kernels can model the design with less data, and hence offer faster optimization.

Most fitness functions will be smooth to a certain degree. An exponential Gaussian kernel which assumes infinite smoothness might not always be accurate. If the fitness function f changes polynomially, which is often the case, the degree of smoothness would ideally be limited. Reconfigurable computing benchmarks will usually be equally smooth over the whole parameter space, apart for few outliers where the underlying design generation changes. This could be possible if a different place and route mechanism was used to map design. The two possible kernel functions which are often appropriate are squared exponential with ARD and Matérn kernel functions, both with additive Gaussian noise. The first offers the automatic parameter relevance determination, yet assumes smoothness. The Matérn kernel is isotropic, while offering a limited degree of smoothness. The Matérn kernel has a parameter ν which is set to either 3 or 5 and regulates the expected smoothness. No benefit from using the Matérn kernel function was observed. Investigation of Matérn kernel with ARD should be considered.

3.4.2 Usability

For the Quadrature-based Financial design comparison to design specific optimizaton tools developed by an expert designer is presented. As previously mentioned the MLO algorithm finishes optimization in as little as 50% of the time taken by the design specific optimization tool, 99 hours vs. 198 hours. This comes at the cost of no guarantee of finding an optimal design, contrary the the expert designed design specific tool. Yet, the example clearly proves the concept of an generic optimization tool. The algorithm can deal with different constraints.

A quantitative comparison is difficult for the PQ design as well as the two designs based on the SMCGen framework. The difficulty with design of a design specific optimization tool for the PQ design comes from timing and PAR constraints. What is possible is construction of a very accurate model of performance of the design. For such a design an expert designer would typically evaluate a couple of configurations using multiple costs tables over a period of a few days. For the SMCGen framework based designs first an expert designer would evaluate a number of designs to establish that mantissa width of the design does not affect its performance. Afterwards, maximum number of cores would be established by analyzing resource reports. Lastly, the designer would set number of particles to satisfy performance while satisfying error constraints. The procedure would take a few days. For all of those designs manual optimization time is estimated between 3-7 days of work. This does mean that potentially an expert designer would be able to

find the optimal designer faster.

The benefit of using the tool comes from offloading the designer from optimization and improving his productivity. There is no guarantee of optimization time for MLO, but the algorithm can deal efficiently with typical reconfigurable design problems. When encountering a design that suffers from PAR or timing problems, the optimization time can increase significantly due to lengthened configuration evaluation time, PQ being such a design. This is no different to manual optimization. Users have to be aware of that, potentially more accurate classifier could mitigate this problem. The MLO algorithm does not scale to designs with more than 3-4 parameters.

3.5 Conclusion

This chapter presents 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 design-specific tool development. The main advantage of using the MLO is a shift from design specific optimization, which can take substantial amount of design time, to automatic computation. The MLO requires multiple benchmarks for further evaluation, and the time that MLO takes is usually substantially lower than the time for design specific optimization with place-and-route and bitstream generation. For one of the designs, MLO finishes optimization in as little as 50% of time taken by the design specific optimization tool. It offers between 3 to 5 times faster optimization than the hill climbing algorithm while offering better performing design configurations.

Recommended algorithm parameter settings are as follows. Infill on highest (lowest) predicted fitness and the squared exponential kernel function with ARD and $min_\sigma = 0.01$ are recommended. The setting was recommended by [146, 66] and our experimental work confirms its good performance. The parameter $\mathfrak{m}$ should be set to values in the range of 10, as in some of the evaluated examples the value drastically lowered optimization performance.

Chapter 4

Design by Efficient Global Optimization

This chapter present the ARDEGO algorithm. It offers a simplified optimization loop compared to the MLO, presented in Chapter 3. The main advantage of this new algorithm is further reduced user input and thus required experience. Furthermore, to speed-up optimization time the algorithm is parallelized. The algorithm takes into account random nature of hardware generation time, the optimization loop is asynchronous. Whenever a configuration is evaluated and a optimization worker node becomes idle, new configuration is prepared for evaluation. The work is published in [88, 47].

FPGAs, and other reconfigurable computing devices, can provide high computational performance but their productive adoption has been hampered due to long hardware design cycles. Describing designs in low-level hardware description languages is more complex than software approaches and the hardware build process is also time-consuming: synthesizing and implementing a single design can take several hours for large modern FPGAs. Advances have been made in high level FPGA design approaches but the problem of long hardware build time remains. In addition to specifying design functionality, a designer is often confronted with multiple non-functional design parameters such as degree of pipelining, memory transfer rates and clock frequency that have to be optimized for performance or power consumption. This optimization requires tremendous effort in analyzing the application to create models and benchmarks, which are then used in the parameter optimization process. Analytical models have been used to tackle this problem for multi-FPGA systems [71]. Numerical representation is known to have significant impact on resource utilization and the number of possible FPGA kernels, and therefore design throughput [155]. Optimization of coefficients of constant multipliers can also yield

improvement [76]. Determining optimal stencil configuration is known to be a difficult problem [109].

Ideally, all those design parameters would be *automatically* optimized. There were several such attempts: Fitness Inheritance was used to decrease number of design evaluations and hence speed-up optimization [117]. In [100] the authors present a design and CAD tool parameter tuning approach. Equally to the design parameters appropriate selection of those CAD tool parameters can allow for generation of better performing design. The technique offers a high degree of parallelization, yet it does not specify how it deals with noise and optimizes only one small design, which takes a few minutes to generate. Timing optimization using cloud computing and machine learning is presented in [80] with a focus on optimization of CAD parameters. In previous work [91, 89] we have demonstrated that it is useful to construct surrogate models of fitness functions representing the design quality of reconfigurable parameterized designs. Yet, creators of a fully automated reconfigurable design optimization tools still face a number of challenges. Challenge one: the algorithm should use simple optimization loop and not require manual setup or tuning for a particular problem. Challenge two: the designers often face multiple constraints, it is common for the majority of the designs in the parameter space to have timing, PAR difficulties or resource *constraints*. Challenge three: the tool should be able to optimize designs based solely on design generation and benchmarking, while being *data efficient*. That is, evaluate as few designs as possible.

We start with treatment of the reconfigurable design parameter optimization problem as a noisy black box global optimization problem. The noisy black box optimization problem is optimization of a noisy function with little or no knowledge of its inner workings. We do this to allow the designer to focus on the actual design, rather then analytical treatment of the optimization problem. To address the two challenges we utilize Bayesian optimization technology inspired by the parallel asynchronous EGO acquisition function [75]. The new ARDEGO algorithm neither requires the designer to tune the algorithm, for example by changing some of its parameters, or to analytically study and model the design. It requires few design evaluations, allowing for an efficient approach. Through introduction of a new adaptive plan it can deal with design, which often violate their constraints. The algorithm is parallelized, and three realistic designs are used for evaluation.

The novel aspects are:

- The novel ARDEGO tool that offers "out-of-the-box" automatic optimization of reconfigurable designs parameters, based on EGO approach. The algorithm combines work on constrained and parallel versions of the algorithm. Furthermore it integrates

a new sampling plan, allowing optimization of designs where only smart part of the parameter space satisfies constraints. Lastly, it integrates the idea of "always valid design" to improve optimization speed. (Section 4.3)

- Due to unkown nature of the underlying problems, ARDEGO performance is largely non-deterministic. To better understand potential bottlenecks and assess performance issues, an analytical assessment of the optimization time is provided. (Section 4.4)

- An evaluation of ARDEGO using three case studies: (a) a quadrature design for financial computation [155], (b) a PQ design [46], and a Reverse Time Migration (RTM) design for seismic imaging with 7 parameters [109]. The algorithm is shown to optimize the RTM design in less than 93 hours, as well as reduction of optimization time of the quadrature-based design from 32 to 25 hours compared to design specific tool for multiple workers, a 22% reduction. (Section 4.5)

- Hardware acceleration of the compute intensive parts of the algorithm is presented. It is shown how idle hardware can be utilized for speed-up of up to 43x of the computation intensive components of the algorithm. (Section 4.6)

4.1 Problem Statement

To allow for parallelization the problem is extended with respect to the problem statement in Section 2.2.2. Evaluation of each parameter setting $f(\mathbf{x})$ involves a certain cost, returned by the cost function $\mathfrak{C} : \mathcal{X} \rightarrow \mathbb{R}^+$. The cost function is random in nature, yet it exhibits some tendencies. For example, the cost increases with resource utilization. In case of parallel optimization algorithms, non-uniform cost implies either an asynchronous problem or large inefficiencies in case of batch parallelization. This becomes prominent when looking at the hardware generation time distribution presented in Figure 4.1. The problem gets further alleviated by the introduction of software design parameters. If the required hardware was already generated parameter evaluation can take as little as few seconds.

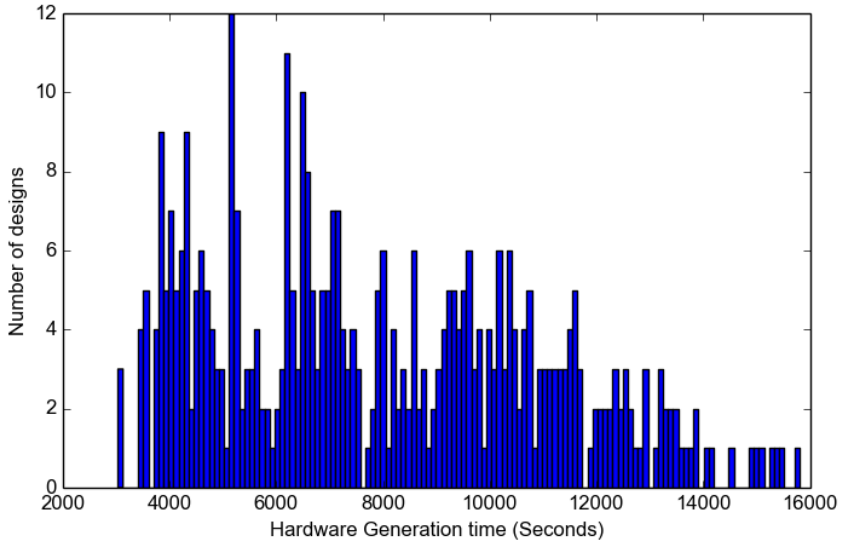


Figure 4.1: Histogram of hardware generation time for the quadrature design [155] implemented using MaxJ for the Maxeler MPC-X1000 system with a Xilinx Virtex-6 XC6VSX475T.

4.2 ARDEGO Approach

The idea behind ARDEGO is to minimize design effort of reconfigurable design optimization. In particular, ARDEGO is applied to problems where multiple parameters have to be optimized to optimize some performance criterion, while possibly being subject to constraints. This is the exact same problem as presented in [89]. There are two obvious approaches to this problem, one is manual tuning and the other is exhaustive search. The first approach is labor intensive and requires a high level of expertise. The second approach evaluates all of the possible designs and as a result is not efficient in terms of the number of performed experiments, even for moderate parameter spaces. Yet, it is fully automated. ARDEGO is used in a similar way. The difference is that the simple loop is replaced with the ARDEGO algorithm.

The key property of the algorithm is its simplicity. Algorithms like MLO presented in Chapter 3 have multiple parameters which can impact their performance. They are based on metaheuristics which exhibit complex behavior [149]. The complex behavior is further aggravated by addition of surrogate models, making their analysis and predictability limited. Metaheuristics can be parallelized [110, 73, 129], and it is also possible to parallelize MLO. Yet, due to the main pitfall of MLO struggling with high dimensional

designs we do not make such attempt. Its optimization loop is already complicated, and to deal with asynchronous optimization we believe that large amount of work would have to be invested in reengineering of its optimization loop. Instead, we follow with ARDEGO.

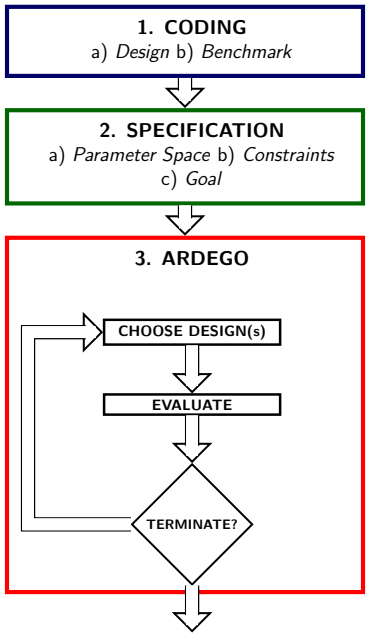


Figure 4.2: ARDEGO optimization approach.

ARDEGO follows a straight-forward optimization loop. It is designed to utilize multiple available cores in an asynchronous fashion, exhibiting large optimization time speed-up. Whenever a worker becomes idle, search for new infill configurations begins. Furthermore, compared to MLO, ARDEGO can deal with high dimensional parameter spaces. MLO algorithm relies on the absolute values of the standard deviation GP prediction from the surrogate model. It uses the prediction to determine when to evaluate a configuration and when to rely on the model. In high dimensional parameter spaces this becomes problematic, the algorithm ends up not using the surrogate model and converges to the standard PSO. Instead, ARDEGO is based on the parallel expected improvement acquisition function [75, 74], and does not suffer from this problem.

This optimization approach has two limitations with respect to using exhaustive search. Currently, only parameter spaces with meaningful distance metrics are supported. Those are spaces, which constitute of a set of values with no underlying meaningful metric. For example, a space with three different Ethernet controllers. The second limitation is that the global optimum is not guaranteed to be found, unlike in exhaustive search. Yet, this is also the case when designer follows manual tuning.

```
# parameter space definition
parameters = {"freq_min" : 100,
              "freq_max" : 100,
              ...
              }

# Build bitstreams and run benchmarks, the fitness function
def buildHardwareRunBenchmark():
      # execute bitstream generation
        os.system("Make_hw")
      # execute benchmark and / or analyze bitstream
        return os.system("Make_run")

# supply the parameter definition and scripts to the optimization algorithm
optimialDesign = ARDEGO(parameters, buildHardwareRunBenchmark)
```

Figure 4.3: The input includes the parameter space specification, and scripts used to generate and benchmark bitstreams.

4.3 ARDEGO Algorithm

The algorithm is designed to offer automatic design parameter optimization. It is based on a GP surrogate model of a reconfigurable design. To account for constraints a SVM classifier is integrated. The key steps of the ARDEGO algorithm are illustrated in Figure 4.4. The algorithm starts to build the initial surrogate model with sampling of the parameter space, generating hardware for these parameter samples and evaluating their fitness. The algorithm invokes P worker nodes, which can build and evaluate configurations in parallel. At any given time μ nodes are busy and λ are idle. After the initial GP and SVM surrogate model is constructed, an iterative process follows where the model is refined with new configurations. The goal is to find a set of λ configurations $\mathbf{X}_\lambda \in \mathcal{X}^\lambda$ that are most likely to improve over the currently best found configuration. This involves global optimization of the new $E[I_{\mathfrak{v}}^{(\mu,\lambda)}(\mathbf{X}_\lambda)]$ acquisition function, inspired by asynchronous parallel EGO and modified for constrained problems. Hardware is then built for the design using infill configurations, fitness is evaluated and the surrogate model updated accordingly. The infill search does not block the optimization on the worker nodes and multiple worker nodes can finish evaluation at similar time. The termination condition is checked whenever a worker node finishes evaluation.

The earlier mentioned challenges are addressed by ARDEGO in the following way:

1. ARDEGO is designed to have a straight-forward optimization loop. It uses an

acquisition function inspired by the asynchronous parallel EGO [75], it uses asynchronous parallelization for faster optimization. An SVM classifier is included and the acquisition function is modified to account for constraints. This is done by setting expected improvement to zero whenever a candidate configuration in $\mathbf{X}_\lambda$ assessed by the acquisition function is predicted to be invalid. The acquisition function presented in [75] fails to proceed with optimization without an integrated classifier. It is a greedy function and is going to continuously evaluate potentially promising, even if clearly invalid, configurations.

2. A classifier allows ARDEGO to prune the parameter space and identify promising regions. The novel *adaptive sampling plan* addresses the issue of parameter spaces with a small number of *valid* configurations.

3. *Data efficiency*: only configurations which optimize the model are evaluated. The experiments are not needed to evaluate acquisition function.

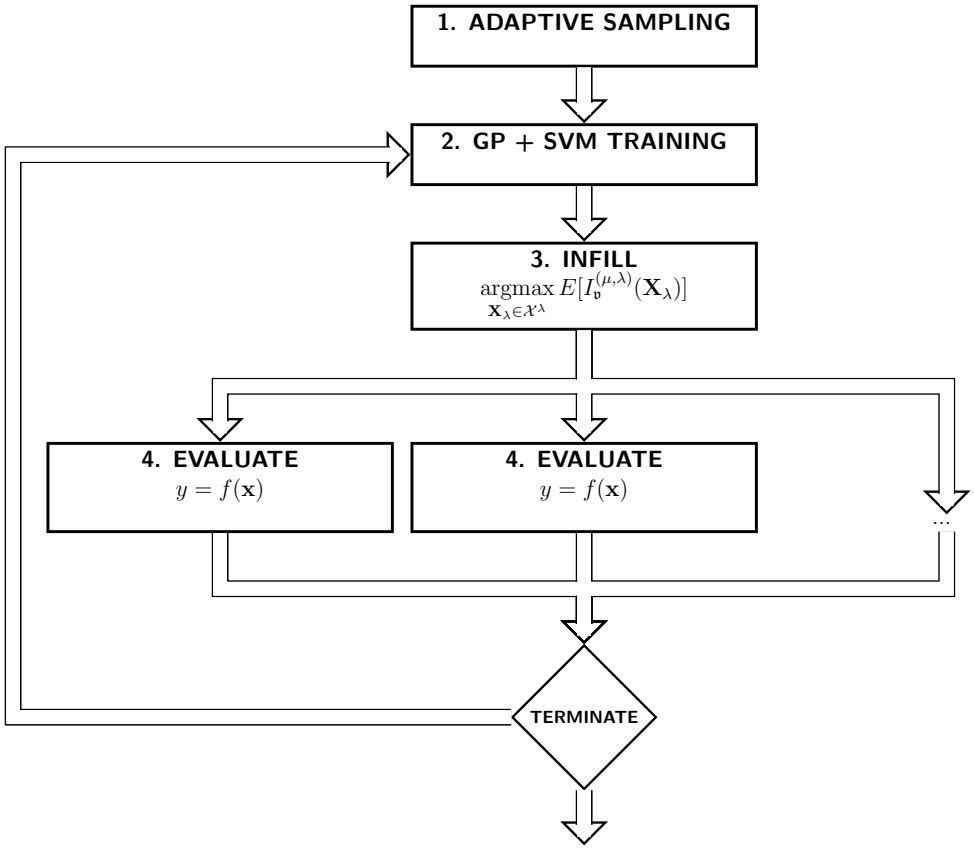


Figure 4.4: ARDEGO, inspired by parallel EGO [75].

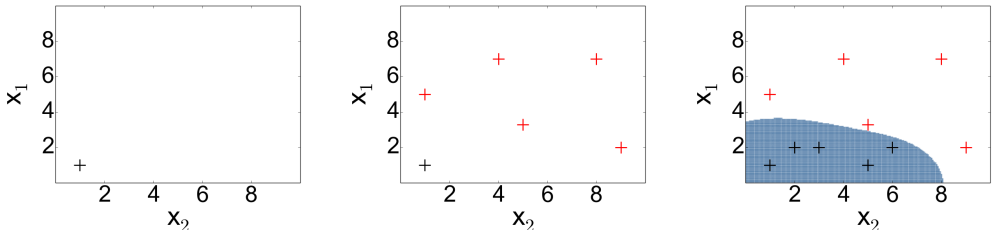
The probabilistic classifiers, like RVMs [154] or GPs for classification [124], are investigated for expensive global optimization in [28, 64]. They are conceptually very attractive replacement for SVMs. However, their integration with a greedy algorithm, like EGO, can be problematic. Weighting of EI by $\mathfrak{p}_{\mathbf{x}}(1)$ is impractical. For example, if the target problem f is an exponential function with decreasing probability of constraint satisfaction; EI will always try to, rightfully, sample the very promising yet unrealistic region of space. The test problems investigated in [28, 64] do not target such functions. The approach followed in this work is to use a non-Bayesian classifier and to identify the valid region $\mathcal{V}$ using SVMs. The goal is to identify the best possible design before exhausting time budget; as the budget is typically very limited the main aim is to find configurations with high $\mathfrak{p}_{\mathbf{x}}(1)$. The belief is that exploring nearby region of valid configurations is sufficient, task for which SVMs are the most adequate.

4.3.1 Adaptive Sampling

The algorithm involves a two-stage *adaptive sampling plan* to allow for a good coverage of the valid space. Initially, if available, an *always valid* configuration is evaluated. This is the configuration, which designer knows works and does not violate constraints. Typically, this would be the configuration, which was evaluated to ensure correctness of the code and scripts. For example, if the parameters are the number of cores and clock frequency this would be the configuration with a single core and lowest possible clock. It is a configuration representing the most basic configuration, which the designer is certain will generate successfully. This allows ARDEGO to localize a region of space, which does not violate any constraints. This is especially important if the valid region is relatively small. The sampling plan has two stages, a Latin hypercube sampling of a number of configurations followed by a random sampling of extra configurations. A Latin hypercube plan offers good space filling qualities, which improve performance of the optimization algorithm [103] while the subsequent random sampling is mainly used to sample more valid configurations for the GP regression. This sampling methodology allows for regression in cases when the valid region is small relative to the parameter space.

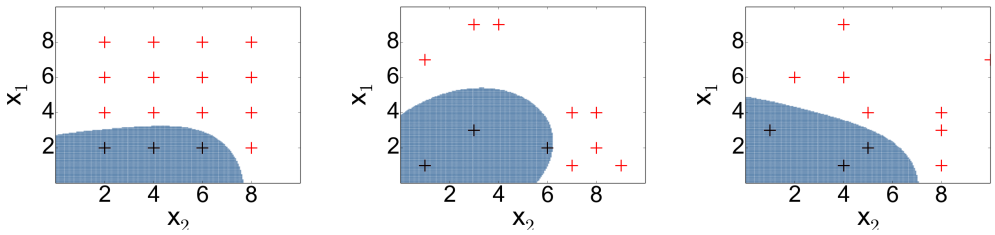
A visualization of the *adaptive sampling plan* is presented in Figure 4.5. Initially the *always valid* configuration is evaluated, followed by a Latin hypercube sampling. The subsequent random sampling plan evaluates a number randomly chosen configurations from the area predicted to be valid by the SVM, tuning its shape. Figure 4.6 illustrates the result of sampling and initial classification in the three other sampling plans. Although

the identified valid areas have similar shape, the number of configurations available for regression is severely limited.



(a) *Always valid* (b) *Latin Hypercube S.* (c) *Random S.*

Figure 4.5: *Adaptive sampling plan*, blue area indicates $\mathcal{V}$.



(a) *Grid* (b) *Random* (c) *Latin Hypercube*

Figure 4.6: Three alternative sampling plans.

4.3.2 GP and SVM Training

After sampling, and whenever new configurations are evaluated, the surrogate model is reconstructed. The training data $\mathbb{X}$ and y is standardized prior to both the GP and SVM training. The surrogate model consists of a GP regressor and SVM classifier. Defined for the purpose of GP training is the set of configurations that resulted in hardware generation and fitness evaluation $\mathbb{X}^+ = \{\mathbf{x}_i | \exists \mathbf{x}_i : [\mathbf{x}_i \in \mathbb{X}] \wedge [t_i \in \mathcal{T}^+]\}$. Given that set, associated design fitness observations $\mathrm{y} = \{y_i | \exists \mathbf{x}_i : [\mathbf{x}_i \in \mathbb{X}^+]\}$ and model hyperparameters θ the GP computes the predictive distribution $\mathrm{p}(f|\mathbf{x}_*, \mathbb{X}^+, \mathrm{y}, \theta)$ for new configurations. The SVM predicts the class $t = d(\mathbf{x}_*)$ for all configurations (i.e. regardless whether the configurations are valid or not) using all observations $\mathbb{X}$ and the observed target labels $\mathbb{t} = \{t_i\}_1^n$. The goal of classification is to construct a decision function d, which allows prediction of class labels $d(\mathbf{x}_*) = t_*$. An illustration of the surrogate model is presented in Figure 4.9.

$L = \text{cholesky}(K(\mathbb{X}^+, \mathbb{X}^+) + \sigma_n^2 \mathbf{I})^{-1})$;
$\mathbf{a} = L^T \setminus (L \setminus \mathbb{y})$;
$\log \text{p}(\mathbb{y}|\mathbb{X}^+) = -\frac{1}{2}\mathbb{y}^T\mathbf{a} - \Sigma_i L_{ii} - \frac{n}{2}\pi$;

Figure 4.7: Mariginal likelihood calculation for Gaussian process regression [124].

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 ARD [35] squared exponential kernel function with additive Gaussian noise, which allows learning of the impact of different parameters on the parameter space [124]. This kernel function has different characteristics across the dimensions (orientations) defined by their respective hyperparameters. The SVM is uses soft-margins [50] to account for a degree of noise and the squared exponential kernel as smooth class boundaries are expected.

$\mathbf{v} = L \setminus \mathbf{k}_*^+$;
$\bar{f}(\mathbf{x}_*) = L \setminus \mathbf{k}_*^+$;
$\text{Var}[\bar{f}(\mathbf{x}_*)] = K(\mathbf{x}_*, \mathbf{x}_*) - \mathbf{v}^T\mathbf{v}$;

Figure 4.8: Prediction of mean $\bar{f}(\mathbf{x}_*)$ and of the variance $\text{Var}[\bar{f}(\mathbf{x}_*)]$ of the estimate using Gaussian process regression [124]. The matrix L and vector $\mathbf{v}$ are computed during model training. The vector $\mathbf{k}_*^+$ is the vector of covariances betwen the configuration and training set $k(\mathbb{X}^+, \mathbf{x}_*)$.

The training of the GP is performed by marginal log-likelihood maximization as presented in Figure 4.7. Due to non-convex nature of the problem, a number of randomly seeded hyperparameter sets as presented in [32]. The hyperparameter set with the lowest negative log-likelihood is chosen. Predictions are done as presented in Figure 4.8.

The SVM is trained using k-fold cross-validation on a grid of hyperparameters C and γ for the squared exponential kernel. The problem is usually solved in the primal-dual relationship [50, 38]. The formulation is for two classes, with labels $t_i \in \{-1, 1\}$. The original problem was

$$\underset{\mathbf{x},\xi,\mathbf{b}}{\text{argmin}} \frac{1}{2}||\mathbf{w}||^2 + C\sum_{i=1}^{n} \xi_i. \tag{4.1}$$

subject to $t_i(\mathbf{w} \cdot \phi(\mathbf{x}_i) + \mathbf{b}) \geq 1 - \xi_i$. Is modified

$$\underset{\alpha}{\operatorname{argmin}}\, \alpha^T Q \alpha - \mathbf{e}^T \alpha. \tag{4.2}$$

where $\mathbf{e}$ is a n long vector of ones, Q is an $n \times n$ positive semidefinite matrix with entries $Q_{ij} \equiv t_i t_j K(\mathbf{x}_i, \mathbf{x}_j)$. The problem is subject to $\mathbb{t}^T \alpha = 0$ and $0 \leq \alpha_i \leq C$ for all training configurations i. The problem is solved only during prediction, when t_i, α, b, C, all nn support vectors and kernel parameter γ are stored and later used during prediction. The prediction for an unevaluated configuration $\mathbf{x}_*$ is performed as follows using the previously stored data

$$d(\mathbf{x}_*) = \operatorname{sgn}(\sum_{1}^{nn} t_i \alpha_i k(\mathbf{x}_i, \mathbf{x}) + b). \tag{4.3}$$

where sgn is the sign function. When there are more then two classes present, "one-against-one" approach is used [112] to compute the decision function.

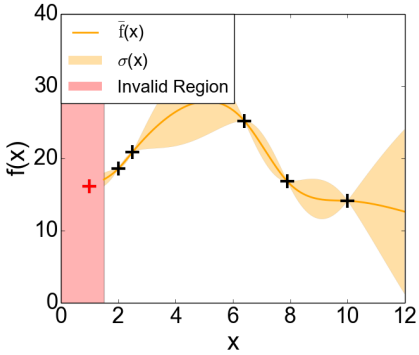


Figure 4.9: ARDEGO surrogate model.

4.3.3 Infill

This step consists of search over the parameter space for λ configurations that are most likely to improve over $\mathbf{x}^+$ and of their subsequent evaluation on the idle worker nodes. In the meanwhile μ configurations $\mathbf{X}_\mu$ are being evaluated on the busy nodes. The algorithm's acquisition function is based on modified asynchronous parallel EGO [75]. Each configuration can take a different amount of time to evaluate, hence the algorithm needs to be asynchronous. Whenever one of the P nodes finishes evaluation, the acquisition function is maximized. If more than one node is free, the acquisition function is used

to find a set of configurations to evaluate on the empty nodes. The surrogate model is updated as soon as a configuration is evaluated, and the configurations which are being evaluated are taken into account in the acquisition function. This ensures that promising regions, which are already being explored, are not overly exploited. The search for the most promising configurations is performed only over the valid space. As $\mathcal{V}$ is not known directly, an SVM classifier is used to predict if a configuration is invalid and modify its impact on the EI. If the configuration $\mathbf{x}_i \in \mathbf{X}_\mu$ is predicted to be invalid, $Y(\mathbf{x}_i)$ is excluded from calculation of EI. This is presented for a set of configurations $\mathbf{X}_\mu$, and the associated GP prediction $\mathbf{Y}^\mu$. A prediction $Y(\mathbf{x}_i)$ is only included in the vector $\mathbf{Y}_\mathfrak{v}^\lambda$ if it is predicted to be valid $d(\mathbf{x}_i) = 1$. The constrained busy configuration vector is defined as follows

$$\mathbf{Y}_\mathfrak{v}^\mu = \{Y(\mathbf{x}_i) | \exists \mathbf{x}_i : [\mathbf{x}_i \in \mathbf{X}_\mu] \wedge [d(\mathbf{x}_i) = 1]\}. \tag{4.4}$$

This is used to create constrained improvement function

$$I_\mathfrak{v}^{(\mu,\lambda)}(\mathbf{X}_\lambda) = \max(0, \max(\mathbf{Y}^\lambda) - \max(f(\mathbf{x}^+), \mathbf{Y}_\mathfrak{v}^\mu)). \tag{4.5}$$

The expectation of the constrained $I_\mathfrak{v}^{(\mu,\lambda)}$ is used to define the acquisition function. Whenever a configuration in the set $\mathbf{X}_\lambda$ is predicted to be invalid $\mathbf{x}_i \in \mathbf{X}_\lambda$, the EI is set to 0. The reasoning is as follows; there are $\lambda - 1$ working nodes left, and there is a candidate set of configurations $\mathbf{X}_{\lambda-1}$ with $E[I^{(\mu,\lambda-1)}(\mathbf{X}_{\lambda-1})]$. The set is extended with a configuration $\mathbf{x}$, predicted to be valid, to produce $\mathbf{X}_\lambda$ and extended with a configuration $\mathbf{x}'$, predicted to be invalid, to produce $\mathbf{X'}_\lambda$. The valid configuration can offer improvement, hence $E[I^{(\mu,\lambda-1)}(\mathbf{X}_\lambda)] \leq E[I^{(\mu,\lambda)}(\mathbf{X}_\lambda)]$. The invalid configuration would bring no contribution to EI, hence $E[I^{(\mu,\lambda-1)}(\mathbf{X}_{\lambda-1})] = E[I^{(\mu,\lambda)}(\mathbf{X'}_\lambda)]$. If there is a choice between $\mathbf{X}_\lambda$ and $\mathbf{X'}_\lambda$, the first is chosen. If there is an invalid configuration in a set, there is an alternative definitely as good and possibly better set. As such, there is no need to calculate EI.

$$EI(\mathbf{X}_\lambda)) = \begin{cases} E[I^{(\mu,\lambda)}(\mathbf{X}_\lambda)], & \forall \mathbf{x}_i \in \mathbf{X}_\lambda, d(\mathbf{x}_i) = 1. \\ 0, & otherwise. \end{cases}$$

and the infill process becomes

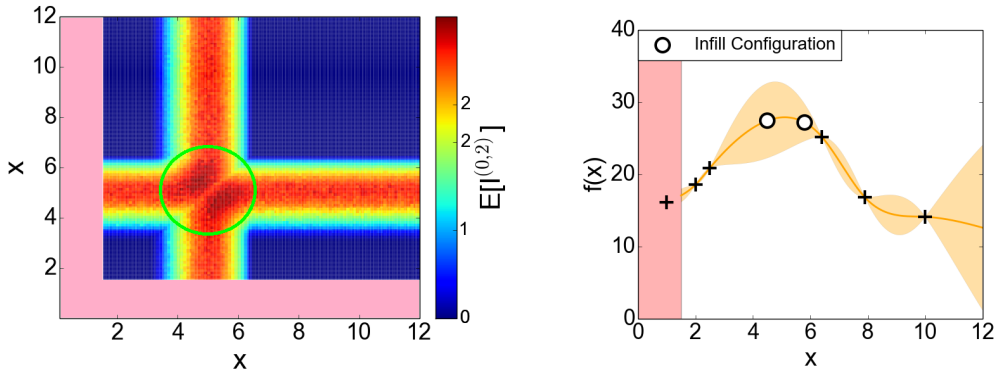
$$\underset{\mathbf{X}_\lambda \in \mathcal{X}^\lambda}{\operatorname{argmax}} E[I_\mathfrak{v}^{(\mu,\lambda)}(\mathbf{X}_\lambda)]. \tag{4.6}$$

As Monte Carlo techniques are used for calculation of EI, the choice between configurations with similar EI can be based purely on the noise in the estimation, even when

a large number of simulations or an adaptive scheme is used [75]. Where as, in that situation ARDEGO is going to evaluate the set of configurations indicated by EI and one configuration in the direct neighborhood of the best found configuration. The $E[I_{\mathfrak{v}}^{(\mu,\lambda)}(\mathbf{X}_\lambda)]$ is chosen as it allows for simpler implementation with less conditional statements and offsets exploitation for exploration around the best found configuration in the later stages of optimization.

There are other possible schemes for integration of classifiers and EGO based optimization. In the case of the sequential EI, it can be multiplied by the probability of a configuration being valid $\mathrm{p}(c(\mathbf{x}_*) = 1|\mathbb{X}, \mathbb{t})$, or a decision boundary can be constructed $\mathrm{p}(c(\mathbf{x}_*) = 1|\mathbb{X}, \mathbb{t}) > 0.5$. The constrained EGO presented in [28] shows such an approach using probabilistic SVMs. RVMs [154] and other probabilistic classifiers such as GPs could be used alternatively, as presented in [64]. Those approaches are not directly suitable for parallelized EI, as the $E[I^{(\mu,\lambda)}$ cannot be directly multiplied by the probability of a configuration being valid, it constitutes of multiple configurations where each has a certain probability of being valid. It is the expected improvement by evaluation of a set of configurations, not a single one. The alternative approach would be to use the probability of individual configurations in $\mathbf{X}_\lambda$ and $\mathbf{X}_\mu$ being valid in calculation of the $I^{(\mu,\lambda)}(\mathbf{X}_\lambda)$. For example, by multiplication of $Y(\mathbf{x}_*)$ by $\mathrm{p}(c(\mathbf{x}_*) = 1|\mathbb{X}, \mathbb{t})$ or by using construction of $E[I_{\mathfrak{v}}^{(\mu,\lambda)}]$ by using threshold values on probabilities of $\mathrm{p}(c(\mathbf{x}_*) = 1|\mathbb{X}, \mathbb{t})$.

The acquisition function maximization is performed using exhaustive search with up to k parameters, and using a PSO algorithm otherwise. This is due to the fact that using exhaustive search to maximize acquisition function could dominate the design optimization time in case of millions of possible configurations in multiple parameters settings. What happens, is that the computation time would become larger than experiment time. Although faster implementations of the acquisition function could allow for exhaustive search with more parameters than k, due to the curse of dimensionality for some D it would still become implausible. That is why a global optimizer has to substitute exhaustive search for some D. The global optimizers computation cost is a fraction of a single configuration evaluation time. This allows for a generous compute budget to maximize the chance of finding the most promising infill. There are other possible acquisition function maximizers [149, 68, 157]). All are plausible, as long as they can deal with non-convex problems. PSO was chosen due to its wide availability and easy implementation. The estimation of $E[I_{\mathfrak{v}}^{(\mu,\lambda)}]$ is presented in Figure 4.11. It consists of the estimation of the fitness and uncertainty, which is subsequently used to calculate $E[I_{\mathfrak{v}}^{(\mu,\lambda)}]$ using Monte Carlo techniques.



(a) *EI Maximization* (b) *Infill*

Figure 4.10: ARDEGO search for infill using $E[I_{\mathfrak{v}}^{(\mu,\lambda)}]$ when two worker nodes are free.

```
// Evaluate only when all configurations in X_λ predicted to be valid
if ∃x_i ∈ X_λ s.t d(x_i)) ≥ 1 return 0;
// Trim the evaluated configurations set X_μ
X_v^μ s.t. ∀x_i ∈ X_v^μ, d(x_i) = 1 ∧ x_i ∈ X_μ;
L =cholesky(Σ_v)) ;
for i ∈ [0, 1, .., q] do
    // Sample the Y vectors
    N ∼ N(0, I);
    Y_v^μ, Y^λ = m(X_v) + LN ;
    // Calculate EI using sampled Ys
    ei += max(max(Y^λ) − max(f(x^+), Y_v^μ));
return ei/(q+1);
```

Figure 4.11: $E_{MC}[I_{\mathfrak{v}}^{(\mu,\lambda)}(\mathbf{X}_\lambda)]$, dervied from [65]. The set $\mathbf{X}_\mathfrak{v}$ is the set $\mathbf{X}$ excluding configurations predicted to be invalid. $\Sigma_\mathfrak{v}$ is the covariance matrix of $\mathbf{Y}_\mathfrak{v}^\mu$ and $\mathbf{Y}^\lambda$.

Due to not always using exhaustive search for maximization of EI, integration of the classifier, discrete space $\mathcal{X}$ and the nature of $E[I_{\mathfrak{v}}^{(\mu,\lambda)}]$ itself it is possible for the new expected improvement function not to indicate any configurations for evaluation. This happens during the later stages of optimization when $E[I_{\mathfrak{v}}^{(\mu,\lambda)}]$ is low and its numerical estimation becomes problematic. Higher number of Monte Carlo simulations can elevate the problem, yet it indicates that non of the configurations offer clearly better performance according to the GP model. It is also possible for $E[I_{\mathfrak{v}}^{(\mu,\lambda)}]$ to indicate some configurations, which have already been evaluated, particularly if the parameter space is discrete. This only happens in the case when P is greater than 1. As an example, $E[I_{\mathfrak{v}}^{(\mu,\lambda)}]$ indicates two configurations one of, which has already been evaluated. The other configuration might

be promising and EI might indicate it for infill while the already evaluated configuration does not contribute towards EI. In this cases, there are three possible solution. The first is to constrain EI to avoid calculation of $E[I_{\mathfrak{v}}^{(\mu,\lambda)}]$ for $\mathbf{X}_\lambda$ for sets of configurations, which contain already evaluated configurations. The other solution is evaluation of some other configuration. The first solution seems promising, yet might stall evaluation in the later stages of optimization. In ARDEGO hill climbing around the best found configuration is used. It allows for a natural transition to a local search, and prevents the algorithm from using EGO methodology when it struggles to discriminate configurations.

4.3.4 Termination

Although the ARDEGO will converge towards an optimum, it is not guaranteed to find the global optimum. Hence, it is crucial to specify termination criteria. There are two possible termination criteria. First, ARDEGO terminates if it exhausts its allocated compute time budget. For example, a number of machines is allocated for a 24-hour period. Alternatively, the ARDEGO algorithm terminates when a parameter configuration $\mathbf{x}$ is found that achieves user required performance.

4.4 Computational Complexity

The two stages of the algorithm are the search for infill configurations and their subsequent evaluation. Design evaluation usually involves hardware generation which often takes hours to complete. However, searching for infill configurations suffers from the curse of dimensionality and theoretically can outgrow hardware generation times for high-dimensional problems. The curse here comes from the fact that the volume of the parameter space increases faster than the number of configurations that can be evaluated. In higher dimensions, more infill points have to be estimated per configuration than in lower dimensions, being potentially time consuming. A model is derived to assess the approximate optimization time during optimization, and hence allow to quantitatively assess potential bottlenecks.

4.4.1 Adaptive Sampling

The time spent during sampling can be split into overhead time and configuration evaluation time. The overhead time, which is the time spent on determining which configurations to sample t_s, is dominated by the SVM training time. It can be assumed to

be $\mathcal{O}(n^3)$ [115]. The training complexity is increased by the two SVM hyperparameters γ and C, to choose the most appropriate values k-fold cross-validation is used over a grid. The complexity of model construction increases with the grid size of g points. The SVMs are retrained during every model construction after the random sampling stage finishes.

$$\mathcal{O}(t_s) = \mathcal{O}(n^3). \tag{4.7}$$

4.4.2 Infill

The main optimization loop has few steps. The time intensive steps are; the surrogate model is update (t_m), acquisition function minimization (t_a) and parameter configuration evaluation (t_e). Each of those steps is repeated during each of iteration. The first step is construction of the *surrogate model*. The construction consists of the SVM classifier and the GP regressor training. The complexity of SVM training and prediction is the same as during adaptive sampling stage. To derive GP regression complexity we assume the limiting case, when all configurations produce fitness $\mathbb{y} = \{y_i\}_{i=1}^{n}$. The GP regression has a $\mathcal{O}(n^3)$ complexity based on the number of evaluated configurations n [124], the complexity is equal to the complexity of SVMs. Training is repeated multiple times to account for the non-convex marginal likelihood function.

$$\mathcal{O}(t_m) = \mathcal{O}(n^3). \tag{4.8}$$

The other time consuming component of the main optimization loop is the acquisition function search $\arg\max_{\mathbf{x}\in\mathcal{X}^\lambda} E[I(\mathbf{x})_{\mathfrak{v}}^{\mu,\lambda}](\mathbf{X}_\lambda)$. The exhaustive search is dependent on the parameter space and is $\mathcal{O}(\mathfrak{m}^D)$ for a space with $\mathfrak{m}$ points per parameter. The parameter $\mathfrak{m}$ will usually be in the range of tens to a hundred. EI assessment of each configuration involves GP and SVM prediction, where the GP prediction complexity dominates with $\mathcal{O}(n^2)$. Therefore, the time per infill using exhaustive search is:

$$\mathcal{O}(t_i) = \mathcal{O}(\mathfrak{m}^D n^2). \tag{4.9}$$

Alternatively a metaheuristic can be used with a budget of $\mathfrak{p}$ acquisition function estimations.

$$\mathcal{O}(t_i) = \mathcal{O}(\mathfrak{p} n^2). \tag{4.10}$$

The final component is evaluation of the infill configurations t_e time. The limiting case is when there is only one worker node, and n configurations to evaluate.

$$\mathcal{O}(t_e) = \mathcal{O}(n). \tag{4.11}$$

4.4.3 Dominant Components

If exhaustive search is used to search for the infill configuration the complexity of t_i is bounded by $\mathcal{O}(\mathfrak{m}^D n^2)$ and becomes an obvious bottleneck. Luckily, despite its highest computational complexity its basic computation cost is orders of magnitude smaller than hardware generation time. Thus, it does not dominate the optimization time, except for large parameter spaces. In those cases a global optimizer is used, like PSO, and under the current budget allocation the complexity is $\mathcal{O}(t_i) = \mathcal{O}(\mathfrak{p} n^2)$. Search for infill is also dependent on the number of Monte Carlo simulations used to estimate EI. The larger the number of simulations, the more accurate the estimate. At the same time t_i increases and can become problematic. Low accuracy of EI can increase n, by less optimal infill choices and thus result in overall longer optimization time. It is crucial to investigate the impact of the number of Monte Carlo estimates on the performance of the algorithm.

The complexity of t_s and t_m is of cubic order, with the number of evaluated configurations n being dominant. Number of evaluated configurations n will unlikely exceed few hundred, meaning training time will not be the bottleneck. Furthermore, both the basic computation time of SVM or GP training is orders of magnitude lower than hardware generation.

4.5 Evaluation

The primary objective of the evaluation section is to compare ARDEGO with alternative techniques. This involves investigation of the impact of the parallelism on its performance. A secondary objective is to determine how the potentially costly Monte Carlo simulations and the new adaptive sampling plan impact ARDEGO, which is compared to Latin hypercube.

Three design benchmarks are used to evaluate ARDEGO optimization time. A quadrature-based financial design with customizable precision [155], a noisy PQ design with custom frequency [46] and a high performance RTM design with seven parameters [109]. All involve complex design choices, and have non trivial constraints. First two

benchmark designs are discussed in Chapter 3. The RTM design is included in the benchmark portfolio to evaluate optimization of designs with larger number of parameters, often encountered in reconfigurable designs. The design is subject to multiple constraints and involves manipulation of stencil configuration.

ARDEGO is compared with surrogate model aided PSO algorithm called MLO presented in Chapter 3 and hill climbing. Hill climbing is a straightforward optimization algorithm, which can deal with mixed continuous and discrete parameter spaces, and makes no assumptions about the fitness function. The MLO algorithm does not offer parallelism, while the hill climbing algorithm can be parallelized. 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 20 experiments. Due to the large parameter space, an analytical model, presented and validated in [109], is used as a substitute for the experiments to mimic the performance of the RTM design. This is tuned using real hardware generations.

The algorithm terminates hardware generation if during the preliminary resource report any of the resources exceeds the FPGA size by more than 10%. This is crucial for automated optimization, the preliminary reports give a good indication of the final resource usage and likely overmapping can be detected as quickly as in 20 minutes.

4.5.1 Implementation

The optimized application designs target a Maxeler MPC-X1000 system with a Xilinx Virtex-6 XC6VSX475T FPGA. The worker nodes consist of high performance Intel Xeon x5650 (32 nm, 6 cores, 2.67GHz) CPUs. The ARDEGO configuration is as follows. The training data is normalized prior to model training. SVM is chosen as the classifier with an squared exponential kernel which is cross-validates on a set of parameters $\gamma \times C = \{1.2^i\}_{i=-10}^{10} \times 10\{1.25^i\}_{i=1}^{10}$. The GP is retrained 100 times with random initialization of the hyperparameters. The sampling stage uses $5D$ configurations for each of the two stages of the adaptive sampling for a total of $10D$, as recommended by [75, 28]. This sampling plan could be problematic for a configuration with a large number of parameters (100+), yet reconfigurable designs rarely have more than a dozen. 5000 Monte Carlo simulations are used for each $E_{MC}[I_{\mathfrak{v}}^{(\mu,\lambda)}(\mathbf{X}_\lambda)]$ estimation. [1]

[1]Experiments were performed with between 5 and 500,000 Monte Carlo simulations, no noticeable impact on the algorithm performance was observed. Although experiments indicated as little as 5 simulations as sufficient, 5000 simulations were chosen as a precautionary measure while not impacting experiment time. In [75] authors suggest using adaptive simulation allocation scheme. The topic requires further study.

Table 4.1: ARDEGO test designs overview.

Design	**Noise**	**Constraints**	**No.**[II]	*D*
Quadrature$^{\delta}$ [155]	Performance of the design exhibits some noise. One of the parameters is a software parameter. Two benchmarks are available, a design throughput and an energy efficiency benchmarks. LUT bound	Accuracy and resource constraints.	20,000	3
PQ$^{\delta}$ [46]	Some of the designs fail randomly due to timing and PAR difficulties. LUT bound.	Resource constraints$^{\omega}$.	8,200	3
RTM† [109]	Not noisy. The valid portion of the design space is small.	Resource and memory bandwidth constraints.	20,160,000	7

II Number of possible configurations in the parameter space.

δ Optimized for Maxeler MPC-X1000 system with a Xilinx Virtex-6 XC6VSX475T FPGA.

† Based on a model tuned using real hardware. The model is set to match Xilinx Virtex-6 XC6VSX475T FPGA.

Table 4.2: The average optimization time in hours and the percentage of the average performance of the optimal configuration of different designs. ARDEGO is compared with various algorithms. The global optimum was found using exhaustive search.

Design	**Quad. Th. (0.1)**	**Quad. Th. (0.01)**	**Quad. Th. (0.001)**	**Quad. En. (0.1)**	**Quad. En. (0.01)**	**Quad. En. (0.001)**	**PQ$^{\text{II}}$**	**RTM**
H.C. $P = 1$	300 (98%)	459 (88%)	393 (72%)	562 (96%)	483 (88%)	**535 (78%)***	903 (93%)	**621 (12%)**†
H.C. $P = 2$	215 (98%)	287 (89%)	220 (74%)	275 (96%)	282 (89%)	251 (75%)	487 (93%)	**241 (12%)**†
H.C. $P = 4$	134 (99%)	146 (88%)	131 (79%)	137 (97%)	147 (88%)	162 (76%)	242 (93%)	**105 (12%)**†
H.C. $P = 8$	75 (100%)	82 (86%)	69 (83%)	82 (97%)	84 (92%)	**69 (69%)**$^{\bullet}$	116 (93%)	**64 (12%)**†
H.C. $P = 16$	46 (100%)	48 (83%)	46 (76%)	58 (93%)	46 (84%)	48 (71%)	62 (93%)	**31 (12%)**†
MLO	182 (100%)	151 (97%)	**190 (95%)**$^{+}$	142 (97%)	102 (100%)	**102 (100%)***	300 (91%)	**N/A**$^{-}$
ARDEGO $P = 1$	130 (100%)	130 (100%)	**106 (100%)**$^{+}$	125 (99%)	103 (100%)	**100 (100%)***	851 (95%)	334 (98%)
ARDEGO $P = 2$	81 (100%)	66 (100%)	64 (100%)	77 (100%)	57 (100%)	64 (100%)	408 (94%)	196 (92%)
ARDEGO $P = 4$	43 (100%)	36 (100%)	37 (100%)	48 (100%)	32 (100%)	40 (100%)	208 (95%)	122 (100%)
ARDEGO $P = 6$	32 (100%)	29 (100%)	29 (100%)	32 (100%)	25 (100%)	28 (100%)	165 (96%)	93 (98%)

1 ARDEGO algorithm uses adaptive sampling plan and 5000 Monte Carlo estimates per EI evaluation.

2 MLO configuration is set as suggested in Chapter 3.

3 The time presented is the average time when an algorithm stopped improving design's performance.

4 Numbers in parenthesis are percentages of the performance of the optimal design configuration.

$+$ ARDEGO can be nearly twice as fast as MLO (when $P = 1$).

$\star$ ARDEGO and MLO can be up to 5 times as fast as the hill climbing algorithm.

$\bullet$ Hill climbing algorithm often struggles with the quadrature-based design, worst case.

$\dagger$ Hill climbing algorithm is not suitable for optimization of designs when number of parameters is large.

$-$ MLO fails to optimize highly dimensional designs.

II Although ARDEGO might look to offer similar performance to other algorithms, the dynamics of optimization seen in Figure 4.14 show its clear superiority.

4.5.2 Quadrature-based Financial Design

In [155] 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. The goal is to find the configuration 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 κ, and the density factor d_f which specifies the density of quadratures used for integral estimation. All of the parameters are uniformly discrete. Energy consumption is measured using Maxeler Technologies provided tools. ARDEGO is evaluated for three different ϵ_{rms}, which influence the number of *valid* configurations. The parameter space $\mathcal{X}$ spans 18,000 configurations, with an average hardware generation time of around 2 hours. The total optimization time using the previous application specific optimization methodology [155], including hardware generation and benchmark execution, is 198 hours. Depending on e_{rms}, the optimal configuration can be between 10-100 times faster and up to 200 times more power efficient than the most basic configuration.

The optimization time of the quadrature design is presented in Table 4.2. The level of parallelism P has a noticeable impact on optimization time. Although it is difficult to make a definite statement due to only 4 different values of P being evaluated, it does not follow the model $speedup = 1 + \mathfrak{b} \log \lambda$ presented in [75]. Least squares logarithmic fitting for the speedup model, $speedup = \mathfrak{a} + \mathfrak{b} \log \lambda$, yields fairly consistent values across different e_{rms} values, where $\mathfrak{a}$ is in the range of 0.75 to 0.9 and $\mathfrak{b}$ in the range of 1.4 to 2.

Contrary to what is expected, the number of simulations does not seem to have any impact on the optimization time. The variance of the results is too high due to relativity low number of experiments, 20 per setting. Even if that is the case, its highly doubtful that the impact would be large. The adaptive sampling exhibits no impact on the optimization time.

Compared to optimization time achieved by the MLO or hill climbing algorithm, as seen in Table 4.2, ARDEGO offers much better performance even without parallelization. Aside of better performance, it nearly always finds the optimal configuration. That is not the case for the two other algorithms. Hill climbing can offer as little as 76% of the optimal configuration performance, while for MLO it is 95%. With regards to optimization time, when $P = 1$ MLO and ARDEGO are comparable for the energy efficiency benchmark. Although increasing parallelism for Hill climbing improves optimization time, clearly the best found configuration deteriorates. In the throughput case, ARDEGO offers a

significant advantage with between 14% and 47% shorter optimization time. When P increases, MLO gets further outclassed.

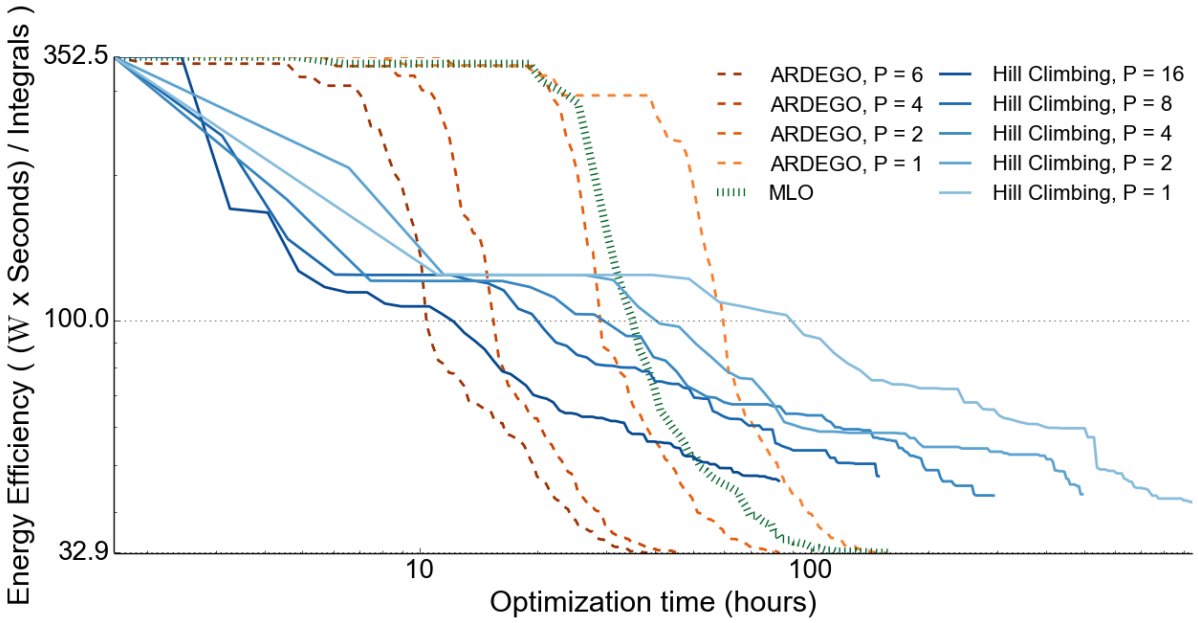


Figure 4.12: Optimization of the quadrature-based financial design energy efficiency benchmark.

The dynamics of optimization are presented for few settings in Figures 4.12-4.13. The speed-up achieved by parallelism is visible in all of the figures. When the number of workers P increases, the curves for both ARDEGO and the hill climbing algorithm are approximately shifted left, indicating equally performing configuration found at an earlier time. Taking into account that optimization time is presented in log scale, it is clearly visible that hill climbing algorithm is at an extreme disadvantage. The hill climbing algorithm takes longer to converge than either ARDEGO or MLO, and rarely finds the optimal configuration. For $P = 1$ MLO offers better configurations at earlier optimization time than ARDEGO, this changes towards the end of optimization.

4.5.3 Real-time Proximity Query Design

Computation of intersections and point-pairs between two objects in 3D has many applications; haptics rendering, computer graphics, virtual prototyping and imaged-guided surgical robotics. To deal with this particular problem authors present the real-time PQ design [46]. The design offers configurable data precision, preserving accuracy while allowing for reduced FPGA resource utilization when needed.

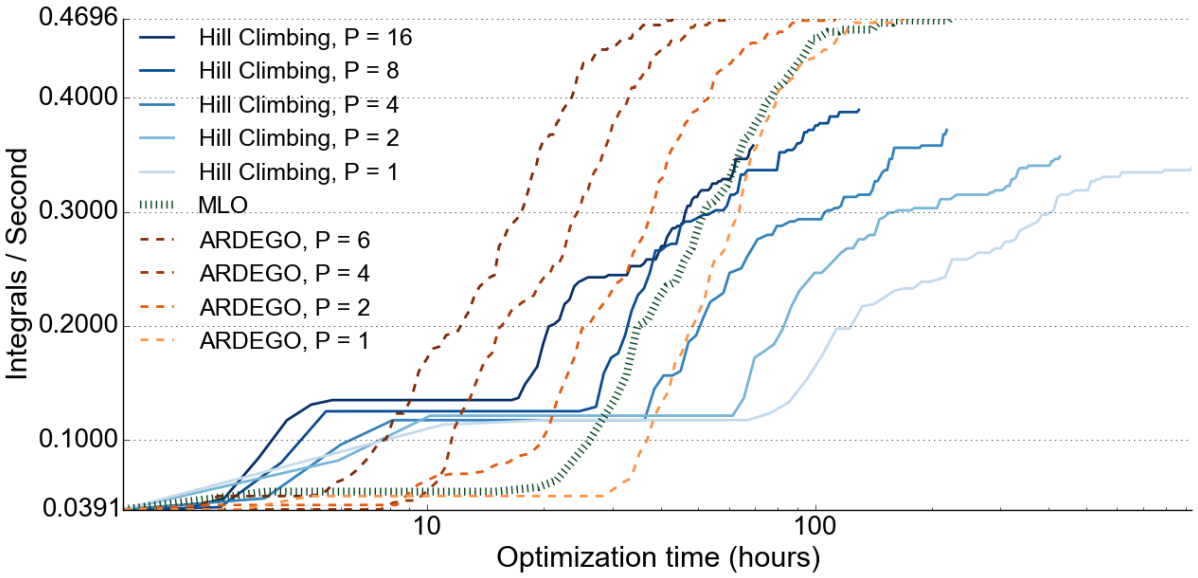


Figure 4.13: Optimization of the quadrature-based financial design throughput benchmark.

The design has three parameters. The mantissa width m_w of the floating point operators, the number of computational cores $\mathfrak{k}$ and the clock frequency of the computational cores $freq$. The mantissa width impacts the number of available computational cores. The higher the numerical precision the more area is required to implement each core. At the same time, higher mantissa width offers higher accuracy and requires less re-computation. The more re-computation is needed, the lower the throughput of the design. Throughput can be increased by higher clock frequency, at the same time it is more difficult for the design to meet the timing requirements. Clearly, the relation between different parameters is complex and optimization is non-trivial. The mantissa width $m_w = [4, 32]$, number of cores $\mathfrak{k} = [1, 4]$, and the frequency $freq = [80, 120]$ MHz. This results in a total of 8200 possible configurations. All of the parameters are uniformly discrete.

Because of clock frequency optimization and timing issues, the design is challenging for optimization. Nearby configurations can fail, and in contrary when logic utilization is an issue, timing prediction is more difficult and generally configuration evaluation cannot not be prematurely terminated. This is clearly visible in the design fitness function visualization presented in Figure 2.7. The design is interesting from the perspective of automatic optimization that uses classifiers, can the algorithms deal with hard to predict invalid spaces. Furthermore, the noisiness of the design generation process makes utilization of the model presented in [46] difficult. The model deals with resource constraints, while the timing is a big issue. This renders the model impractical for optimization purpose

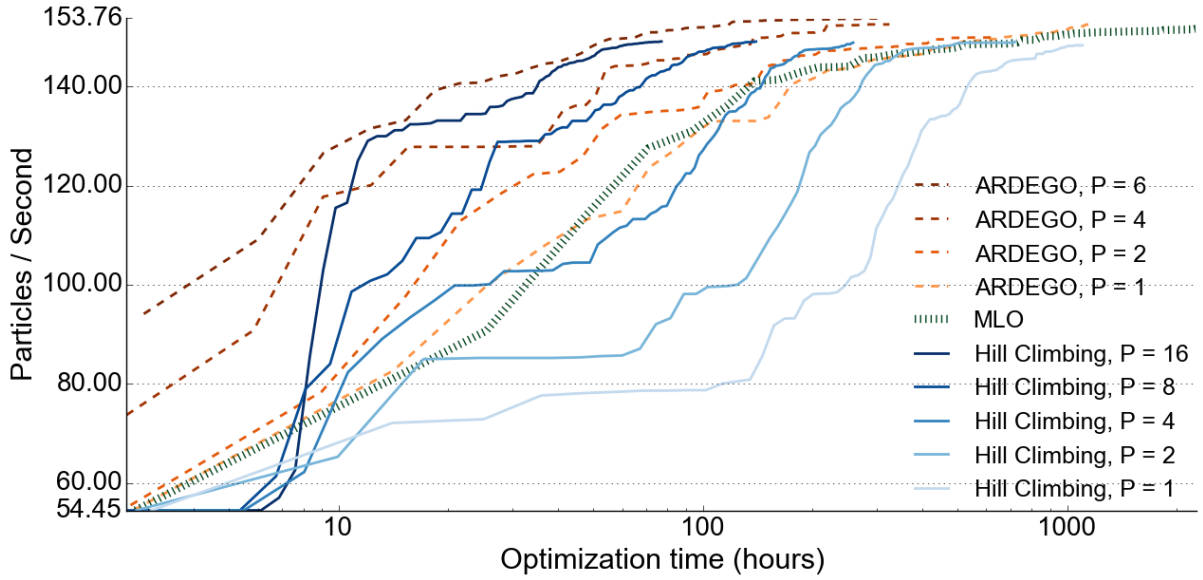


Figure 4.14: Optimization of the PQ design throughput.

and calls for other methods of optimization.

The dynamics of optimization are presented in Figure 4.14. It is clear that ARDEGO offers superior performance to both MLO and the hill climbing algorithm. For P=1, MLO and ARDEGO offer similar optimization time, yet hill climbing algorithm requires two to four nodes to match them. When increasing P, both ARDEGO and the hill climbing algorithm improve their performance. The hill climbing algorithm requires 16 nodes to offer similar performance to ARDEGO with 6 nodes, yet is still inferior. The design is challenging for MLO and ARDEGO due to noisy parameter space and classifier struggling to properly classify the parameter space. The problems resulting from PAR and timing constraints are largely random in nature, although probability of successful bitstream does vary with the designs parameters. An SVM classifier constructs a hard decision boundary, failing to accommodate that information. The design is challenging for the hill climbing algorithm for a different reason. Although the optimal configuration is located relatively closely to the starting point, *the valid configuration*, initial navigation through the noisy space is time consuming. The Table 4.2 shows that all of the algorithms struggle to find the optimal configuration, and usually terminate when their configuration evaluation budget has been exhausted.

The trends visible in the optimization of the PQ design are similar to ones presented for the quadrature design. Increasing parallelism P offers a clear speed-up, again of logarithmic nature. The results are presented in Table 4.2. Equally number of Monte

Carlo simulations seem to have no clear impact on the optimization time, although the results are not presented and the adaptive sampling does not offer any benefit with respect to Latin hypercube. With regards to the two alternative algorithms, MLO and hill climbing, ARDEGO offers a clear improvement.

4.5.4 Reverse Time Migration Design

In [109] the designer faces a problem of optimizing seven parameters of a high performance RTM design and there are around 20 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 configuration. The parameters explored are blocking ratios in two 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. All of the parameters are uniformly discrete. Hardware generation time takes up to 9 hours for a single configuration. An analytical model has been developed to enable optimization of the design involving memory architectures, precision optimization, computation transformation, and design scalability [109]. The optimized design is over 100 times faster than the basic configuration, the unoptimized basic design.

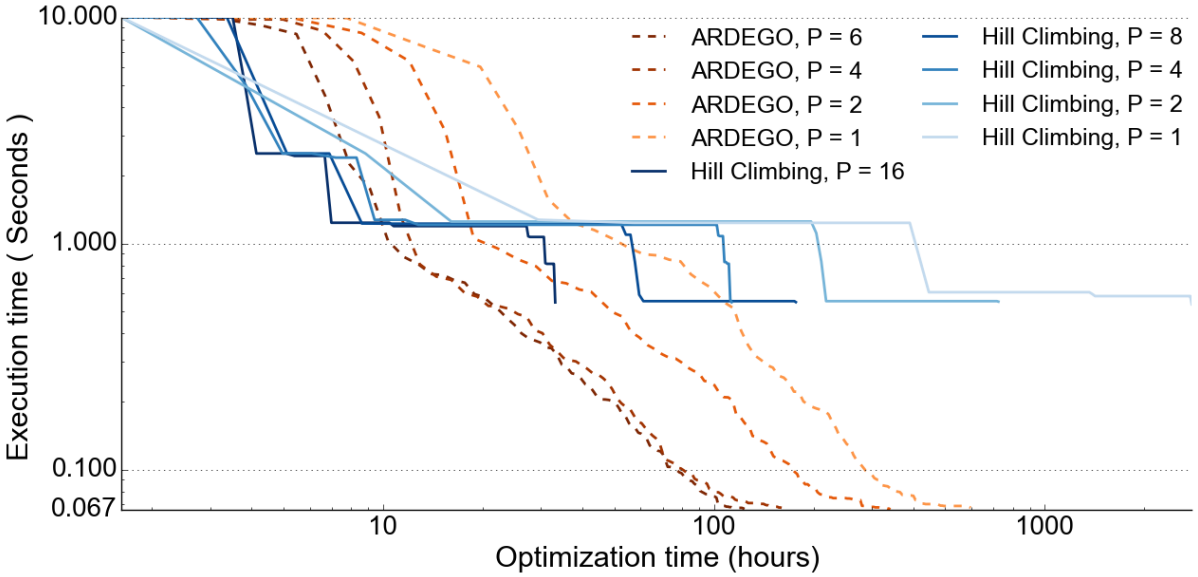


Figure 4.15: Optimization of the RTM design execution time.

The trends visible in the optimization of the RTM design follow similar pattern as in previous designs. The level of parallelism P greatly improves performance of the optimization achieving logarithmic speed-up. The number of Monte Carlo simulations seems to have slight impact on the optimization time when $P = 2$. This is probably the case as increasing P does not yield more accurate results, yet it increases the computational burden of Monte Carlo simulations. This is not clear and can be due to low number of experiments.

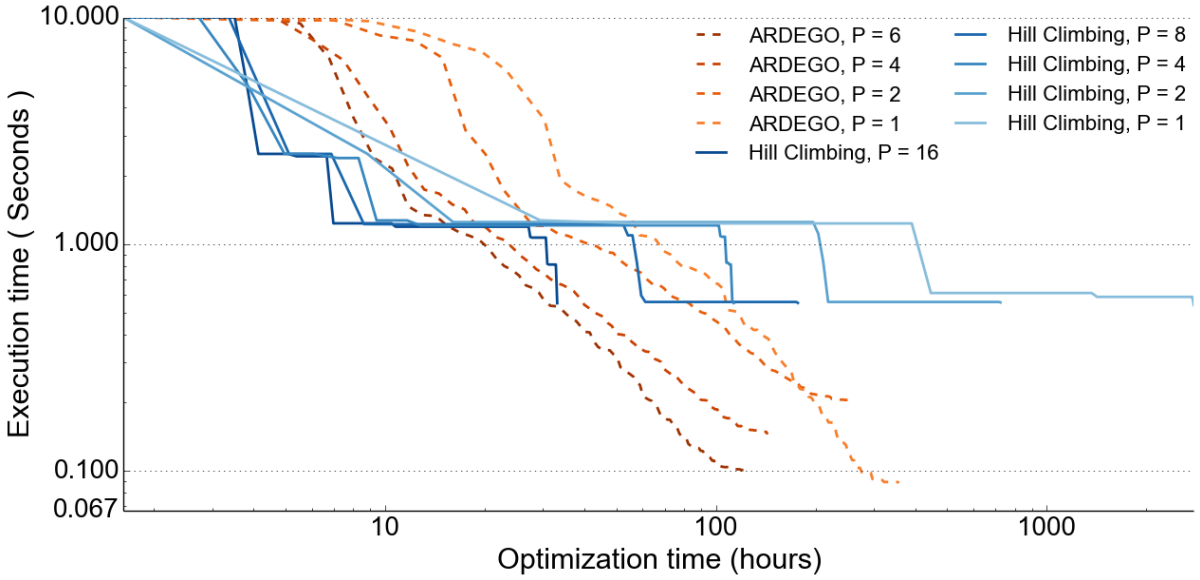


Figure 4.16: Optimization of the RTM design execution time, Latin hypercube sampling.

Contrary to the previous examples, the adaptive sampling plan yields massive improvements to the optimization time. This is the case as valid area includes only 3.2% of the available parameter space. Using Latin hypercube sampling instead of the adaptive sampling plan, ARDEGO finds configurations offering between one third and three quarters of the optimal configuration performance. This is regardless of P and the number of Monte Carlo simulations.

Optimization of the RTM design was not possible using MLO algorithm, hence, it is left out of the comparison. Due to its reliance on absolute standard deviation output from the surrogate model, for higher dimensional problems the MLO algorithm ends up evaluating all configurations instead of using surrogate model estimates. The EI metric relies on relative value and as a result ARDEGO does not suffer from this problem. The hill-climbing algorithm offers poor performance, mainly due to the highly dimensional nature of the problem. It does not get close to the optimal configuration, and terminates

at around 1000 optimization hours, against 100-200 for ARDEGO. This is presented in Figure 4.15.

4.6 Hardware Acceleration

The most computationally demanding component of ARDEGO are Monte Carlo estimates of $E_{MC}[I_{\mathfrak{v}}^{(\mu,\lambda)}]$. During every infill procedure at least one worker, and this typically means at least one FPGA, are idle. This means that there is a potential for acceleration of ARDEGO. The FPGA circuit should allow for higher number of Monte Carlo estimates at no additional cost, the hardware is idle and present in the system.

The $E_{MC}[I_{\mathfrak{v}}^{(\mu,\lambda)}]$ estimation problem has no data hazards and easily maps onto streaming computing model. The code from Figure 4.11 is parallelized. For maximum throughput the design is heavily pipelined, and C-slowing technique is applied [93]. Depending on the depth of C-slowing buffer, a different number of points are estimated in a batch. The circuit is presented in Figure 4.17. The surrogate model prediction and the accumulation and division are done in software. The surrogate model prediction and accumulation and division are done only once per $E_{MC}[I^{(\mu,\lambda)}]$ calculation and would introduce significant resource cost to the design.

The design is multi-core, with each core consisting of a memory and data-path elements. Memory element stores the data required to create random number generators Y^{μ} and Y^{λ}. Three BRAM elements labeled with "BRAM L", "BRAM μ" or "BRAM λ_i" store the decomposed convariance matrix, and means of the associated $\mathbf{Y}_{(\omega)}^{\mu}$ or $\mathbf{Y}_{(\omega)}^{\lambda}$ vectors. The memory element also contains register storing $f_{\mathbf{x}^+}$ value. The memory can be shared across numerous data-paths, effectively becoming unrolling factor. Each data-path generates a Gaussian random number for every Y^{μ} and Y^{λ}, using the data supplied from memory. The rest of the data-path performs the calculation as defined in $E_{MC}[I^{(\mu,\lambda)}]$. The results from each data-path are aggregated in the C-slowing buffer and returned from the engine after a programmed number of Monte Carlo simulations.

The maximum μ and λ limit is configurable, allowing for custom level of parallelism P. The design has multiple number of *data-paths* per number of *cores*, increasing the throughput and resource cost. BRAM memory components are reused across data-paths if overmapping on BRAM. The ratio of memory components to data-paths has to be balanced as it can cause difficulties during place and route process. In the current implementation the Gaussian random number generators follows [151]. It is a low resource implementation with quality sufficient for billions of samples. As a rule of thumb, the generator used in

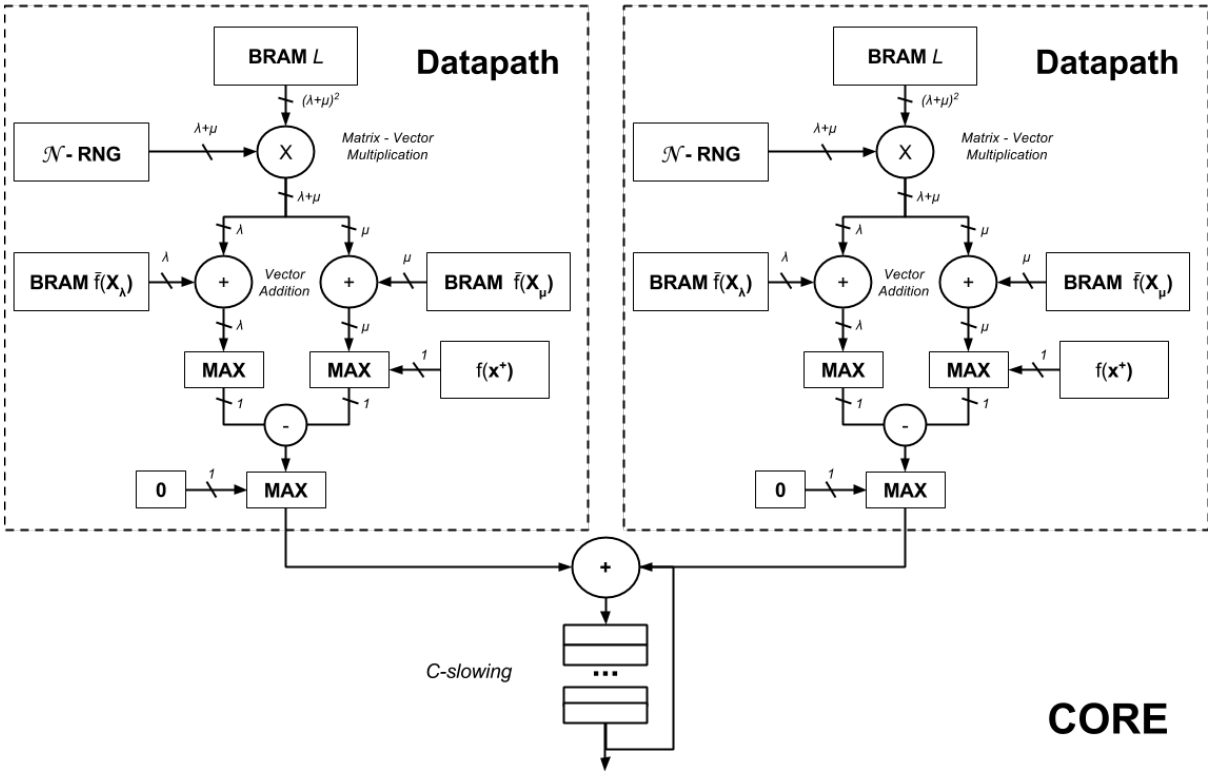


Figure 4.17: A circuit used for acceleration of the $E[I^{(\mu,\lambda)}(\mathbf{X}_\lambda)]$ and $E[I_{\mathfrak{v}}^{(\mu,\lambda)}(\mathbf{X}_\lambda)]$ functions. The connectors widths represent the number of double or floating point numbers.

the estimation of $E_{MC}[I^{(\mu,\lambda)}]$ has to retain statistical quality for at least the number of used simulations. Once this criterion is achieved, resource utilization can be optimized to maximize throughput and include extra cores and data-paths.

The circuit is applicable both to $E_{MC}[I^{(\mu,\lambda)}]$ and $E_{MC}[I_{\mathfrak{v}}^{(\mu,\lambda)}]$. In the case of $E_{MC}[I_{\mathfrak{v}}^{(\mu,\lambda)}]$ the simulation simply not proceeds if and of the infill parameter settings is predicted to be invalid. If any of the already being evaluated parameter settings is predicted to be invalid, its standard deviation is set to 0. This effectively transforms $\mathbf{Y}^\mu$ into $\mathbf{Y}_{\mathfrak{v}}^\mu$ and allows the same circuit to handle all cases. An alternative is to reconfigure FPGA with a different configuration to accommodate smaller vector $\mathbf{Y}_{\mathfrak{v}}^\mu$ and use the spare resources to increase the number of cores and data-paths.

Hardware Evaluation

The circuit is compared against a high performance Intel Xeon x5650 CPU runs software implementation of the $E_{MC}[I^{(\mu,\lambda)}]$. The CPU runs at 2.67GHz utilizing all 6 cores. Software was compiled with CPU specific optimization flags for maximum performance.

Abramowitz and Stegun method [20] is used for software Gaussian random number generation, although Marsaglia [101] and other were investigated. The method offers good performance and reasonable quality random numbers. The hardware is configured with 2 cores, each with unrolling factor of 4 and one memory unit. The maximum λ and μ are set to 6, allowing estimations of $E_{MC}[I^{(\mu,\lambda)}]$ with μ and λ between 0 and 6. Double precision arithmetic is used for both software and hardware. The clock frequency is 150 MHz. C-slowing is set to 100. Throughput of the FPGA and software implementation is presented in Figure 4.18.

Figure 4.19 shows the speedup figure of FPGA solution over software. The speedup is assessed by varying λ between 1 and 6, both μ and λ have equal impact on the cost of the integral estimation. It becomes apparent that speedup increases with λ, or P, and the number of Monte Carlo simulations. Speedup improves with λ as the software solution becomes slower as λ increases, while FPGA throughput is constant. The maximum observed speedup was 43x over our software implementation when $\lambda = 6$. For low number of Monte Carlo simulations the FPGA on-chip memory unit takes relatively large amount of time, yet becomes insignificant as the number of simulation increases. Although [75] uses as little as 1000 simulations for initial $E_{MC}[I^{(\mu,\lambda)}]$ estimations, higher number significantly increases the estimate accuracy. Finally, during later stage of optimization when high number of Monte Carlo simulations becomes beneficial, the accelerated $E_{MC}[I^{(\mu,\lambda)}]$ engine offers high speedup.

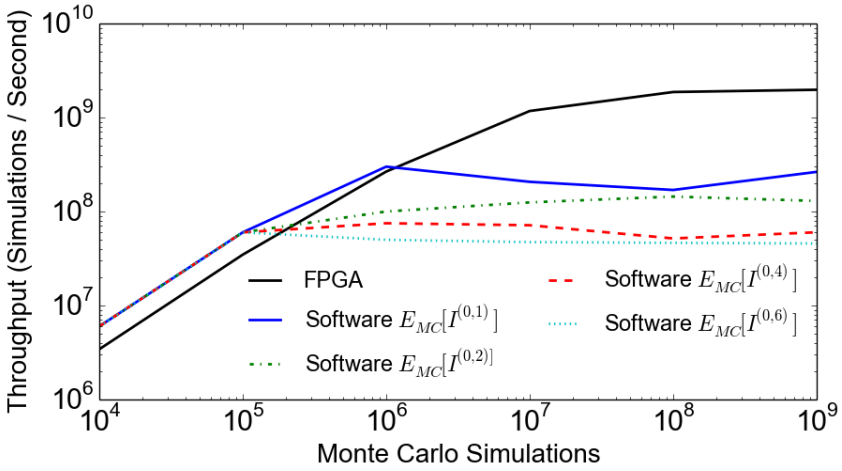


Figure 4.18: Throughput of unoptimized hardware and multi-core software implementations of $E_{MC}[I^{(\mu,\lambda)}]$.

The computation cost is solely dominated by the Gaussian random number generation. Assuming software being able to generate 20 million Gaussian random numbers per second

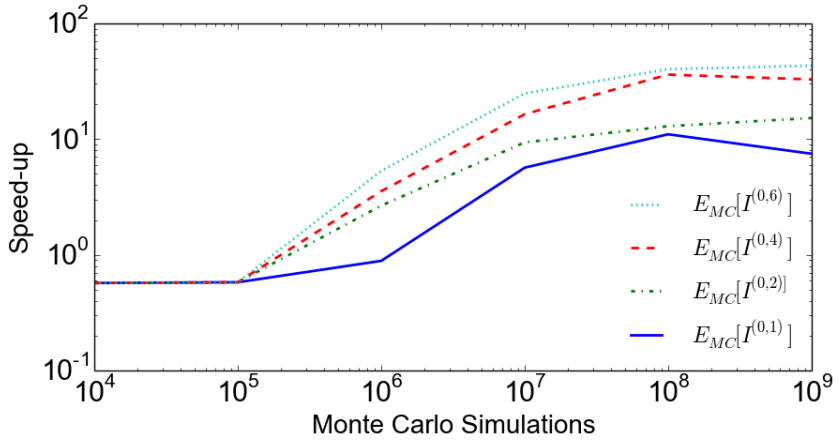


Figure 4.19: Speed-up of the hardware vs. the software solution with increasning number of Monte Carlo simulations.

per 1 GHz of clock speed [152] the throughput of a 6 core 3 GHz CPU is

$$\text{Software Throughput} = 6 \times 3 \times 20,000,000 = 360,000,000. \tag{4.12}$$

360 million random Gaussian numbers per second. The throughput of the FPGA solution is number of cores $\mathfrak{k}$, times the number of data-paths $\mathfrak{d}$, times the clock frequency $\mathfrak{f}$, finally times the supported level of parallelism $\mu + \lambda$. This gives

$$\text{Theoertical FPGA Throughput} = \mathfrak{k} \times \mathfrak{d} \times \mathfrak{f} \times (\mu + \lambda). \tag{4.13}$$

which in the current implementation gives

$$\text{FPGA Throughput} = 2 \times 4 \times 150 \text{ MHz} \times (6 + 6). \tag{4.14}$$

a total of 14.4 billion random Gaussian numbers per second. This gives a theoretical speed-up of 40x, matching the experimental results of 43x.

Software implementation could be further optimized, for example using SSE instruction. Yet, a all of the CPU 6 cores were utilized it becomes apparent that even with extreme optimization FPGA solution offers huge performance benefit. It has to be taken into account that the FPGA engine configuration is not optimized, and substantial speed-up

is expected. Different random number generators, evaluation of custom arithmetic among many possible improvements. Most importantly, the FPGA is part of the system and is idle, hence not using it would be wasteful. Currently GP and SVM predictions are the dominant components in $E_{MC}[I^{(\mu,\lambda)}]$ calculation, unless for extremely large number of simulations.

4.7 Discussion

4.7.1 Results

The algorithm is not substantially faster than MLO when $P = 1$. Yet, it has a simple optimization loop and offers parallelism. This results in as much as 85% optimization time reduction when using 6 workers, 190 vs. 29 hours. The design specific tool requires 198 hours to finish optimization, reduced down to 33 hours assuming 6 worker nodes and perfect parallelization. ARDEGO requires at most 32 hours to finish optimization of quadrature-based design regardless of error constraints. In the best cases it requires just 25 hours, a 22% reduction in optimization time.

MLO cannot optimize designs with multiple parameters, like the RTM design. The ARDEGO algorithm is shown to optimize the RTM design in less than 93 hours, and hill climbing algorithm fails to proceed with optimization due to dimensionality. With 6 worker nodes for the quadrature-based design ARDEGO offers up to 46% reduction of optimization time compared to the hill climbing algorithm with 16 nodes, 25 vs. 46 hours. Furthermore, hill climbing algorithm does not always find the optimal design. Summarizing, compared to the hill climbing algorithm, ARDEGO offers better performing configurations and uses less computing power resulting in an much increased efficiency. This is the direct result of data efficiency, only configurations, which either are promising or resign in unexplored region are evaluated. Hill climbing explores the parameter space in an inefficient manner, hoping that the next configuration will offer better performance. It offers speed-up with increased level of parallelism, yet it often struggles with optimization and cannot be applied in higher dimensions.

The adaptive sampling plan offers similar performance to Latin hypercube sampling plan for low dimensionality when the valid region constitutes large portion of the parameter space. In the RTM case where the valid region covers only 2.2% of the parameter space, the benefit of the new sampling plan becomes noticeable. Is allows the algorithm to find configurations offering twice the performance of the best configuration found when using

Latin hypercube sampling plan.

The number of Monte Carlo simulations has no clear impact on ARDEGOs optimization performance. Although not presented, the effect of the number of Monte Carlo simulations on the performance of the algorithm was investigated, and no impact was observed. As little as 5 simulations, and as many as 500,000 were investigated. We believe that currently more accurate estimations do not offer a clear benefit due to one of the two reasons.

First, the ARDEGO algorithm requires more accurate classification mechanism and other components have a much smaller impact. The PQ design has a noisy design space, where hardware generations often fail due to PAR and timing issues. The RTM design has multiple constraints and resides in a large parameter space. Quadrature design has multiple constraints, the optimal configuration is difficult to locate. The algorithm often samples invalid designs, something that is clearly wasteful.

Second, the number of simulations would need to be further increased, beyond 500000 per point, and more thorough search is needed to observe any benefit. The hardware acceleration allows for more thorough evaluation of the parameter space, and allows for larger parameter spaces to use exhaustive search for acquisition function maximization during infill. It is possible that using exhaustive search for acquisition function maximization for highly dimensional designs, like RTM, would bring positive impact. Due to increased search space with λ, exhaustive search should always be considered infeasible for λ>1. Software solution, with throughput of up to 100,000,000 simulations per second, is limited in what it offers. Disregarding GP and SVM prediction cost it would take around 15 minutes to perform exhaustive infill search for the RTM design with 5000 simulations per configuration when $\lambda = 1$. With ARDEGO taking around 150 iterations to finish optimization, this adds up to a 40 hour overhead. Further work on acceleration is worth an investigation. As at least one idle FPGA is available during infill, it seems wasteful not to use it.

4.7.2 Usability

For the Quadrature-based Financial design comparison to design specific optimizaton tools developed by an expert user is presented. The ARDEGO algorithm offers better performance than a parallelized design specific optimization tool, for 6 worker nodes a reduction of up to 22%. As in the MLO case, the optimization comes at the cost of no guarantee of finding an optimal design.

The major improvement compare to MLO in terms of algorithms usability comes

from its parallelization and ability to deal with designs with larger number of parameters. Parallelization allows the algorithm to mitigate long hardware generation time. Parallelism is widely employed by expert users, evaluating multiple design configuration in parallel. The ability to deal with larger number of parameters allows the algorithm to be applied to a much wider range of designs. For example, the author of the RTM design, clearly an expert designer, spent over a week just to construct analytical models of the design's performance. This time does not include subsequent hardware generations. Clearly ARDEGO could substantially improve his productivity. At the same time construction of those models allows designers to better understand the underlying problem.

For a novice designer the ability of the algorithm to deal with a larger number of parameters and offering parallelism could simply make reconfigurable hardware feasible. Often designers want to utilize reconfigurable designs only for a particular problem and can not justify the required time investment to both learn a HDL and inner workings of reconfigurable designs. By tighter coupling of ARDEGO with tool chains the design cycle could be simplified automating optimization.

4.8 Conclusion

We present ARDEGO, an asynchronous parallel algorithm for automatic optimization of design parameters in reconfigurable designs. ARDEGO is evaluated using three case studies, a quadrature-based financial application, a proximity query design used in 3D object intersection detection and an reverse time migration design for seismic imaging. All designs show a clear time saving when using ARDEGO over hill climbing and the MLO optimization algorithm, which was presented in Chapter 3. Up to 85% reduction of optimization time is observed with respect to the MLO algorithm and a 22% compared to design specific tool. Optimizing the RTM design, we show that ARDEGO can optimize highly-dimensional designs. When using 6 worker nodes, the algorithm manages to finish the optimization in as little as 93 hours. The previously presented MLO algorithm could not optimize the 7 dimensional design. Simple techniques, like hill climbing, can optimize designs with few parameters; however, they fail to deal with larger number of parameters due to their data inefficiency and exponentially growing search spaces.

Although the ARDEGO algorithm can still take substantial amount of time to finish optimization, it offers a clear advantage over other approaches as well as manual approach which require extensive designer interaction to study and tweak the design. The ARDEGO algorithm allows for parallelism, offering logarithmic speed-up, greatly reducing

optimization time.

Chapter 5

Design and Knowledge Transfer

It has been previously shown how optimization of reconfigurable designs can be automated. Example of designs that can benefit from such schemes are numerous: A reconfigurable radio throughput is highly dependent on the level of parallelism [30], by balancing the numerical representation and re-computation designs power, throughput and accuracy can be optimized [155, 46], whilst stencil configuration has non-obvious impact on designs performance [109], multiplier constant co-efficients can have substantial impact on design performance [76].

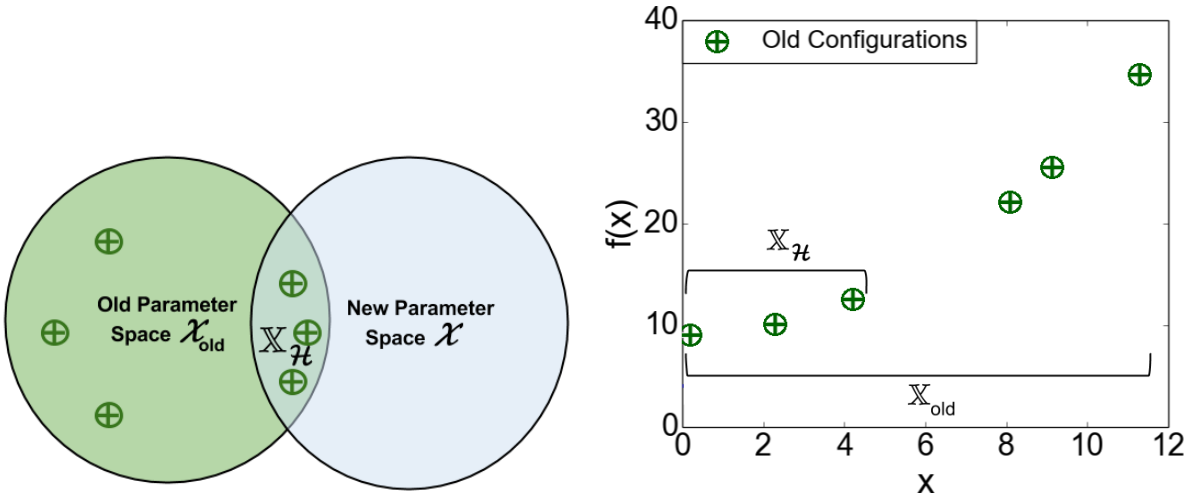
Previous research on automated optimization of reconfigurable designs involves both design and CAD tool parameters. Cloud computing and machine learning can be used to tune CAD parameters for faster optimization [80]. In [100] optimization of CAD tools and design parameters is presented, although only a small design is used for evaluation. In Chapter 4 we show how Bayesian optimization methodology is used to treat noise, offering a simple algorithm and allowing for parallelism to speed-up optimization time. The key advantage of the Bayesian optimization methodology, is that it is data efficient. It relies on the model of a design, and uses experiments through hardware generation to refine it. However, all of the approaches are wasteful. When an optimization finishes, a number of bitstreams are discarded, when they could be used to navigate algorithms in future optimization attempts. The key idea presented in this chapter is how to prevent this type of waste, and to show how previous optimization attempts can yield faster optimization.

We present the concept of recovery of lost knowledge during previous design optimization, and how to benefit from it. The key idea is to store information like bitstreams and benchmark output during optimization in view of later reusing it to optimize future designs. Currently, designers discard this information. The work presents the Auto-Transfer algorithm; it extends the ARDEGO algorithm presented in Chapter 4 with a

new knowledge transfer step. The concept of knowledge transfer is related to the idea of transfer learning [113]. Transfer learning is a new field in machine learning, which tackles the problem of transferring knowledge from one problem onto another. It deals mainly with three different tasks: (a) "what to transfer" between the tasks, (b) "when to transfer" and (b) "how to transfer". The difference between this approach and transfer learning, is that we only transfer the previously collected data to speed-up new design optimization. In transfer learning, the knowledge gained about the problem, such as optimized hyperparameters, is transferred as well.

We identified two use cases. The first use case: designs are often ported across platforms and it is crucial to use bitstreams generated for one platform to speed-up optimization of the design being ported onto a different platform. The second use case: designs with related architectures often offer similar performance behaviour, and this knowledge can be of great value especially in the context of frameworks, like [24, 47]. This use case includes using old bitstreams to speed-up optimization of a revised design. This is a common situation in the reconfigurable computing community. At times, once the design is completed, it gets tested and optimized and only then new features are requested or a bug is discovered. Knowledge transfer is easy for a human being, but becomes a challenge in an automated approach. We approach the problem by utilizing Bayesian optimization, in particular the new Auto-Transfer algorithm.

- Statement of the problem of knowledge transfer in reconfigurable design optimization. We formally define the problem of reusing old results of reconfigurable design optimization to yield optimization of new and revised designs. (Section 5.1)
- Identification of two different knowledge transfer cases and their treatment in the context of automatic reconfigurable design optimization. (Section 5.2)
- Presentation of an algorithm for automatic knowledge transfer in the reconfigurable design optimization context, the Auto-Transfer algorithm. It derives from ARDEGO presented in Chapter 4. Instead of initially randomly sampling the parameter space, it uses knowledge stored in the form of previous design synthesis and benchmarking results. (Section 5.3)
- An evaluation of Auto-Transfer algorithm using three case studies: The first case study is a quadrature design for financial computation [155]. It is built for two different platforms and we show how bitstreams generated for an old platform can be used to guide the algorithm for faster optimization of the design on a next generation platform. We follow with a similar treatment of the RTM design for seismic imaging



(a) *Common parameters* (b) *Relationship between functions*

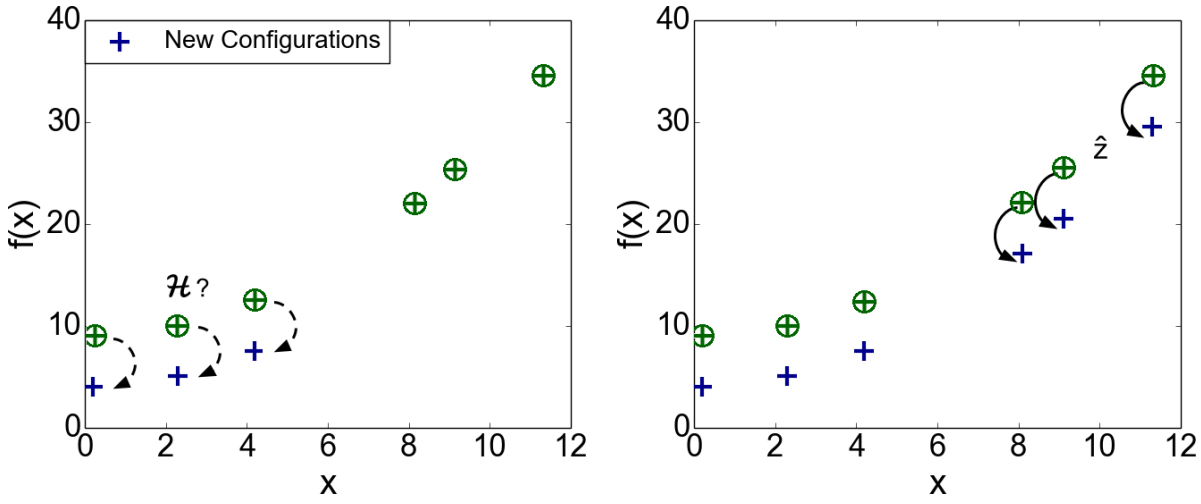
Figure 5.1: In (a) six parameter settings are in the old design respository. The set $\mathbb{X}_{\mathcal{H}} \in \mathbb{X}_{old}$ consists of three out of six of those configurations. The three configurations can be evaluated for the new design and used to verify the relationship between the two designs.

with multiple parameters [109]. Lastly, we attempt to reuse hardware generated between two related designs. A stochastic volatility design and a robot localization design [47]. By using knowledge transfer optimization time is reduced by up to 35% compared to ARDEGO. (Section 5.4)

5.1 Problem Statement

To allow for knowledge transfer the problem is extended with respect to the problem statement in Section 2.2.2. The user often faces the problem of creating a design similar to a previously created one. The designer applies design patterns or uses his prior experience to improve his efficiency when creating the new design. The same process can be replicated in automated optimization. There are two designs, an *old* design and a new one. The knowledge, which includes the information learned and the old data, can be used to speed up the new design optimization. The problem we are concerned with is how to transfer the old data. There are three challenges: (a) "what to transfer", (b) "when to transfer" and (c) "how to transfer" [113].

The old design has associated fitness functions f_{old}, as well as multiple constraint functions $h_{old,i}$ and $g_{old,j}$. Typically the following data is aggregated in the repository: $\mathbb{X}_{old}$ evaluated parameter settings, y_{old} associated finesses and exit codes indicating failed



(a) $\mathbb{X}_{\mathcal{H}}$ *evaluated* (b) $f(\mathbf{x})$ *mapped using* z

Figure 5.2: To transfer the old parameter settings, there needs to be a set of parameters $\mathbb{X}_{\mathcal{H}}$, which can be used to verify the hypothesis that there is a relationship between the designs. In this case the set consists of parameter settings $\mathbf{x}_1$ to $\mathbf{x}_3$. Those parameter settings allow to identify the relationship between the old and the new design. This is later used for more accurate modeling and optimization.

constraints $\mathbb{t}_{old}$. All of that information can be potentially transferred. In particular, it is possible to transfer knowledge if there are some easily detectable function z s.t. $z_f \circ f_{old} = f$ and/or z s.t. $z_i \circ h_{old,i} = h_i$ or $z_j \circ g_{old,j} = g_j$. Aside of expert knowledge, the only way to detect if there are any relationship, and what they are, is to evaluate a set of the same parameter configurations. This also means that the parameter space of the old design $\mathcal{X}_{old}$ and of the new design $\mathcal{X}$ needs to have a non-empty common set $\mathcal{X}_{old} \cap \mathcal{X} \neq \emptyset$. It is also possible to revise the parameter space. The concept is presented in Figure 5.1. For example, when moving from Xilinx to Altera platforms the width of mantissa and exponent of floating point operators follow different restrictions.

If those relationships are discovered, some of the parameter settings from the old repository can be used for treatment of the new problem, as shown in Figure 5.2. This should allow for better modeling of the new design, thus resulting in a decreased number of new parameter setting evaluations and faster optimization. The knowledge transfer of the old results $\mathbb{X}_{old}$, $\mathbb{y}_{old}$ and $\mathbb{t}_{old}$ is only beneficial when functions z can be cheaply identified. The optimization time saving due to knowledge transfer should be greater than the cost associated with using it. Otherwise, transfer knowledge becomes pointless. The concept is called "negative transfer" in transfer learning and cannot be excluded [113].

5.2 Auto-Transfer Approach

The goal is to recover any useful information collected during old design optimization to decrease the required number of parameter configuration evaluations for the newly optimized design. The task of selecting an old design from the database suitable for knowledge transfer is not currently automated. In the current work the designer indicates a design which he believes is related to the new design. There are three challenges related to knowledge transfer: (a) "what to transfer", (b) "when to transfer" and (c) "how to transfer" [113]. The new approach is outlined in Figure 5.3. It is based on the new Auto-Transfer algorithm.

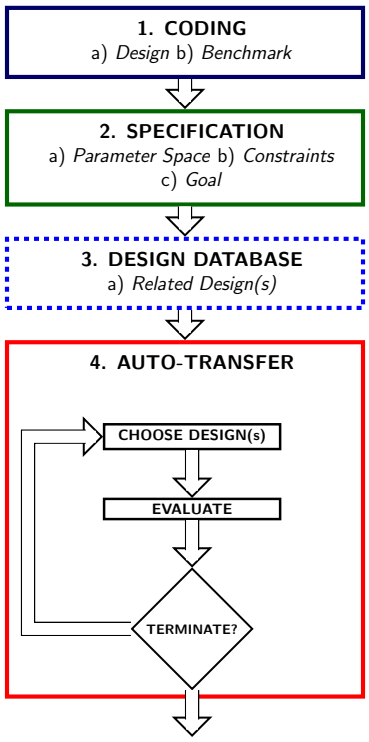


Figure 5.3: Knowledge transfer optimization approach, extended to accomodate old designs.

The Auto-Transfer algorithm starts with the knowledge transfer step; the designer provides an old design database. Ideally the database would contain data of previous design optimization and a feature set allowing to identify individual designs and configurations. This would allow an algorithm to automatically find the most similar designs and transfer the data gathered during their optimization. Unfortunately, creation of such a meaningful feature set requires a lot of data, which is currently not available. Due to the cost of hardware generation it is possible that such a feature set will never be constructed,

although cloud computing might make it possible in the future. The database used in this approach is specified in the Subsection 5.3.1 and addresses challenge (a). The database could contain data gathered during optimization of the same design for an older platform. It could also contain data gathered during optimization of a related design which shares code, and hence some characteristics, with the new design. The designer is responsible for providing a database containing optimization results of a similar design. Challenge (b) is addressed by testing of different hypothesis to verify if data can be transferred. The Auto-Transfer algorithm verifies the relationship through evaluation of some of the parameter configurations of the new design. This set of parameter configuration is used for hypothesis testing. It is described in Subsection 5.3.2. Then, depending on the hypothesis testing outcome, the data will or will not be transferred. This addresses challenge (c) and is described in Subsection 5.3.3. It helps to minimize the possible negative impact of knowledge application. Two candidate use cases are identified for the Auto-Transfer algorithm.

The **cross-platform transfer** is a common case, when a design is ported onto a new platform. The new platform offers different amount of resources, has different topology and timing characteristics.

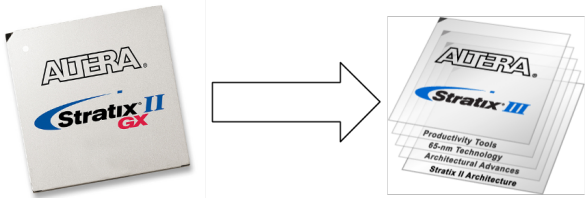


Figure 5.4: Moving to a new platform [6].

The second use case is **related designs**. The designs can be based on the same core algorithm or framework and the designer believes that they follow similar behavior. Examples of such frameworks are [47, 94]. The approach is suitable in cases when the designs follow the same parameterization, or there is a straight-forward mapping between them, which the user can indicate. Designs that are revised to improve performance or to include extra features fall into this category.

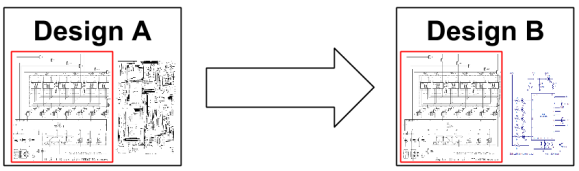


Figure 5.5: Related designs.

```
1
2  # design database
3  design_db = {[100, 1, 11, 53] : [55.0], # Previously optimized designs
4               [100, 4, 8,  53] : [22.0], # and data gathered during their
5               ...                        # optimization is supplied to the
6              }                           # algorithm.
7
8  # parameter space definition
9  parameters = {...}
10
11 # Build bitstreams and run benchmarks, the fitness function
12 def buildHardwareRunBenchmark():
13       # execute bitstream generation
14         os.system("Make hw")
15       # execute benchmark and / or analyze bitstream
16         return os.system("Make run")
17
18 # supply the parameter definition and scripts to the optimization algorithm
19 optimialDesign = ARDEGO(parameters, buildHardwareRunBenchmark, design_db)
```

Figure 5.6: The algorithm input is extended with a design database. The design database stores results of previous optimizations, allowing ARDEGO algorithm to transfer knowledge. This can improve both the speed and accuracy of the optimization.

The current approach is distinct to transfer learning [113] in that it does not reuse previously learned information. For example, in the case of GPs prior can be modified to accommodate previously learned information or previously most promising kernel function can be used [124]. For example, one could discourage equal weighting of different parameters by using anisotropic squared exponential kernel function and a prior which penalizes equal magnitude length-scale hyperparameters. The Auto-Transfer algorithm was not extended to follow such approach due to time constraints.

5.3 Auto-Transfer Algorithm

The approach is based on the ARDEGO algorithm presented in Chapter 4. The Auto-Transfer algorithm is part of a wider class of Bayesian optimization algorithms. It relies on a GP surrogate model, which drives the optimization. It starts with knowledge transfer step after, which it proceeds to infill. Infill is a procedure of finding promising parameter settings for evaluations, based on the surrogate model. The infill evaluates the most promising settings as indicated by the acquisition function. The Auto-Transfer algorithm uses the $E[I_{\mathfrak{v}}^{(\mu,\lambda)}(\mathbf{X}_\lambda)]$ acquisition function presented in Chapter 4. It accounts for constraints and allows for asynchronous parallelism during optimization.

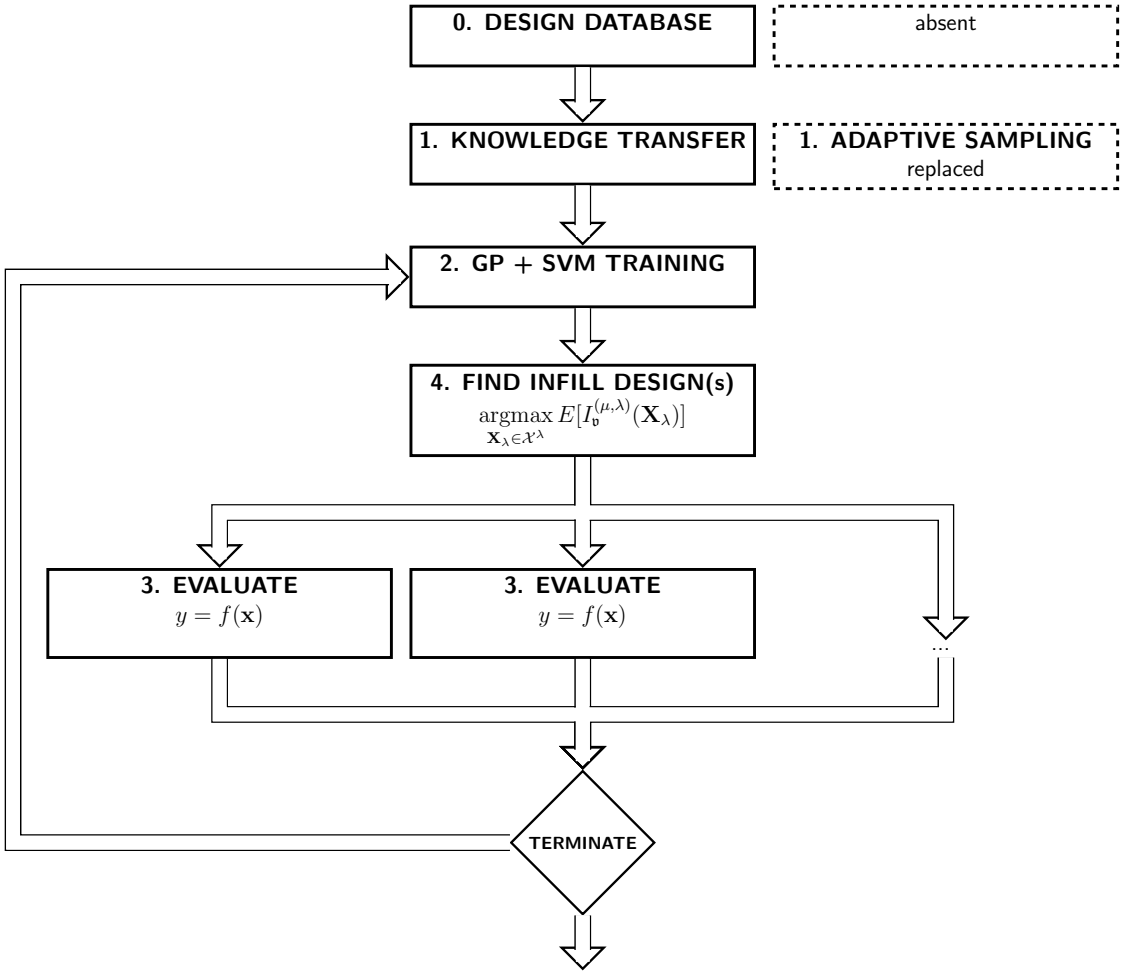


Figure 5.7: The Auto-Transfer algorithm with comparison to the ARDEGO algorithm.

The Auto-Transfer algorithm does not initially sample the design space, unlike the ARDEGO algorithm or the MLO algorithm presented in Chapter 3. Instead, it uses a *knowledge transfer step* as presented in Figure 5.7. The step relies on user-provided data, which contains data gathered during optimization of an old design. Relationship between the old design's fitness and constraint functions is recovered by evaluation of a subset of old parameter settings $\mathbb{X}_{\mathcal{H}}$ provided in the database. Then, Pearson (linear) [114] or Spearman (monotonic) [167] correlation estimators are used to tests the hypothesis that there are either linear or monotonic relationships between the old and the new design. Conceptually, using information theoretic approach seems very attractive. For example, mutual information [69] can be used to measure mutual dependence between between the fitness functions and the constraint functions using the old data and new data evaluated for the configurations $\mathbb{X}_{\mathcal{H}}$. Directional relationship can be identified using transfer entropy

[128] or symbolic transfer entropy [145]. Unfortunately, in general, information theory requires very large number of samples to provide statistically significant results, making it impractical in the current context. Based on the observations in Chapter 4, n_{old} is going to vary between 30 and 150 samples, making its subset $\mathbb{X}_{\mathcal{H}}$ even smaller. Typically, this can be considered to be smaller then the sample size D 10 used for sampling in Chapter 4. On the other hand, although with certain restrictions, Pearson (linear) [114] or Spearman (monotonic) [167] correlation estimators can be used to identify significant very strong correlations using small number of samples [37].

If a relationship is discovered, a regression model is constructed. The model is used to map the information gathered in the database and treat the old experiments as if they came from the new design. The benefit comes when the cost of testing of those relationships is smaller than the time saved by constructing more accurate models and thus speeding up optimization.

5.3.1 Knowledge Transfer

The process of knowledge transfer has three aspects, each addresses one of the previously mentioned challenges. First, a set of parameter settings is evaluated for the new platform. Secondly, the users hypothesis that the designs are related is tested. Lastly, if the hypothesis holds, the knowledge accumulated in the database is transferred. This is done by using the uncovered functions z_i to map the fitness calculated for the old design, as well as other information, onto the new design.

Challenge (a) – What To Transfer

The database consists of previous n_{old} parameter sets $\{\mathbf{x}_i\}_{i=1}^{n_{old}} = \mathbb{X}_{old}$, the associated performance figures for which hardware was successfully generated $\forall \mathbf{x}_i \in \mathbb{X}_{old}$ s.t.$f_{old}(\mathbf{x}_i) = y_i \rightarrow y_i \in \mathbb{y}_{old}$, as well as exit codes $\{t_i\}_{i=1}^{n_{old}} = \mathbb{t}_{old}$. Whenever possible, the database contains $k + r$ vectors $\mathbb{v}$ associated with all of the k equalities and r inequalities, s.t $\forall \mathbf{x}_i \in \mathbb{X}_{old}$ s.t.$h_{old,i}(\mathbf{x}_i) = v_j \rightarrow v_j \in \mathbb{v}_{old,i}$. Equally, in the case of inequalities $\forall \mathbf{x}_i \in \mathbb{X}_{old}$ s.t.$g_{old,i}(\mathbf{x}_i) = v_j \rightarrow v_j \in \mathbb{v}_{old,i}$. An example of a constraint, the LUT usage has to be lower than the number of LUTs available. Another example is accuracy. Throughput constraint is also plausible, the design is only accepted after it reaches a certain threshold. All of those values are recorded. If any of the constraints fails, the exit code is recorded as well. An example is presented in Table 5.1.

The goal is to identify the relationship between the old and the new design using as

Table 5.1: An example of a design repository $\mathbb{X}_{old}$ of a design configurable with m_w mantissa width of floating point numerical operators and the number of cores *cores*. A total of 8 parameter settings were evaluated.

m_w	*cores*	Throughput $\mathbb{y}_{old}$	Accuracy $\mathbb{v}_{old,1}$	Latency $\mathbb{v}_{old,2}$	LUTs $\mathbb{v}_{old,3}$	Exit code $\mathbb{t}$
32	1	102.5	0.001	1 ms	35%	0
19	3	305.6	0.005	3 ms	44%	0
35	5	n/a	n/a	n/a	n/a	1
19	5	507.2	0.005	6 ms	72%	0
21	5	506.5	0.004	6 ms	81%	0
24	5	502.4	0.002	6 ms	89%	0
15	7	703.1	0.1	9 ms	92%	2

Table 5.2: An example $\mathbb{X}_{\mathcal{H}}$ for the design presented in Table 5.1 and a related design. There is a clear difference in performance.

m_w	*cores*	Throughput $\mathbb{y}$	Accuracy $\mathbb{v}_1$	Latency $\mathbb{v}_2$	LUTs $\mathbb{v}_3$	Exit code $\mathbb{t}$
32	1	201.2	0.001	1 ms	25%	0
19	3	405.3	0.005	7 ms	24%	0
21	5	606.8	0.004	9 ms	57%	0

few design evaluations as possible. There are two relatively easily verifiable relationships. The algorithm allows for either linear or monotonic relationships between the old and new f, h_i and g_j. The process starts by reevaluation of a random subset of $n_{\mathcal{H}}$ parameter settings $\mathbb{X}_{\mathcal{H}}$, which are present in $\mathbb{X}_{old}$; they can be evaluated on the new platform and involved successful hardware generation. This subset is used for hypothesis testing. If there is no such subset, then knowledge transfer cannot proceed.

Challenge (b) – When to Transfer

First a test is performed to verify if there is a linear relationship between any of the old and new functions f, h_i and g_i. For example, a design has been ported from a Xilinx to an Altera FPGA. The test verifies if there is a linear relationship between the throughput of the design on the previous f_{old} and the new platform f. Then, in the case of comparison of fitness functions, $\forall y_i \in \mathbb{y}_{old}, \exists \mathbf{x}_i \in \mathbb{X}_{\mathcal{H}}$ s.t.$f_{old}(\mathbf{x}_i) = y_i$ and $\forall y_i \in \mathbb{y}, \exists \mathbf{x}_i \in \mathbb{X}_{\mathcal{H}}$ s.t.$f(\mathbf{x}_i) = y_i$. The size of old and new $\mathbb{y}$ vectors does not have to be the same. It is possible for some of the designs evaluated on the old platform to violate some constraints on the new platform, or vice versa. If any of the function f, h_i and g_j

cannot be evaluated for the new platform, they are excluded from their associated $\mathbb{y}$ or $\mathbb{v}$ vectors. To test the linear relationship hypothesis $\mathcal{H}_l$, the Pearson product-moment correlation coefficient [114] is calculated between those sets.

$$\rho_{(\mathbb{v}_{old},\mathbb{v})} = \frac{\mathrm{cov}(\mathbb{v}_{old}, \mathbb{v})}{\sigma_{\mathbb{v}_{old}}\sigma_{\mathbb{v}}} \tag{5.1}$$

The Pearson product-moment is equally performed to test for a linear relationship between fitness functions using vectors $\mathbb{y}$ and $\mathbb{y}_{old}$.

The test is only performed if the set $\mathbb{X}_{\mathcal{H}}$ contains at least three designs, otherwise the Pearson product-moment correlation coefficient and associated hypothesis testing will always indicate a linear relationship between two sets of values. Although the sample size is typically going to small, taking into account that the user indicated related designs, the former should be sufficient. With a low sample size, due to the cost associated with design evaluation, the significance level for the double sided p-value is restrictive. It is important to note that the vectors $\mathbb{v}$ can be of different sizes for the old and the new design, and for different constraints. For example, if the design was built for the old platform but overmaps on resources on the new platform, its fitness cannot be assessed.

Table 5.3: Correlations calculated for the set $\mathbb{X}_{\mathcal{H}}$ presented in Table 5.1 and Table 5.2. With $alpha = 0.05\%$ the $\mathcal{H}_l$ is rejected for the latency, although the $\mathcal{H}_m$ holds. For mapping of the LUTs, both hypothesis are rejected.

m_w	$cores$		Throughput		Accuracy		Latency		LUTs	
			$\mathbb{y}_{old}$	$\mathbb{y}$	$\mathbb{v}_{old,1}$	$\mathbb{v}_1$	$\mathbb{v}_{old,2}$	$\mathbb{v}_2$	$\mathbb{v}_{old,3}$	$\mathbb{v}_3$
32	1		102.5	201.2	0.001	0.001	1 ms	1 ms	35%	25%
19	3		305.6	405.3	0.005	0.005	3 ms	7 ms	44%	24%
21	5		507.2	606.8	0.004	0.004	6 ms	9 ms	81%	57%
Pearson correlation (p)			1.0 (0.0%)		1.0 (0.0%)		**0.92 (25%)**		**0.97 (15%)**	
Spearman correlation (p)			0.99 (0.035%)		1.0 (0.0%)		1.0 (0.0%)		**0.87 (33.0%)**	

If the linear relationship hypothesis $\mathcal{H}_l$ is rejected at an α significance level or indicates a weak correlation, a test is performed to check the hypothesis $\mathcal{H}_m$ that there is a monotonic relationship between the two functions. Again, the test is only performed if the set $\mathbb{X}_{\mathcal{H}}$ contains at least three designs. To test the hypothesis $\mathcal{H}_m$ the Spearman rank correlation is calculated [167]. The test values $v_{old,i}$ and v_i are converted into ranks $\mathfrak{r}_{old}$ and $\mathfrak{r}$. The Spearman rank correlation is then defined as

$$\rho_{(\mathbb{v}_{old},\mathbb{v})} = 1 - \frac{6\sum_{i=1}^{n_{\mathcal{H}}} d_i^2}{n_{\mathcal{H}}^3 - n_{\mathcal{H}}} \tag{5.2}$$

where $d_i = \mathfrak{r}_{old,\,i} - \mathfrak{r}_i$ is the difference between ranks. Similarly to the Pearson correlation, double sided p-value is calculated. If the $\mathcal{H}_m$ is rejected at an α significance level or there is a weak correlation, it is assumed that the null hypothesis $\mathcal{H}_0$ holds and that there is no relation between the two tested functions. The Spearman rank correlation is equally calculated to test for a monotonic relationship between fitness functions using vectors $\mathbb{y}$ and $\mathbb{y}_{old}$.

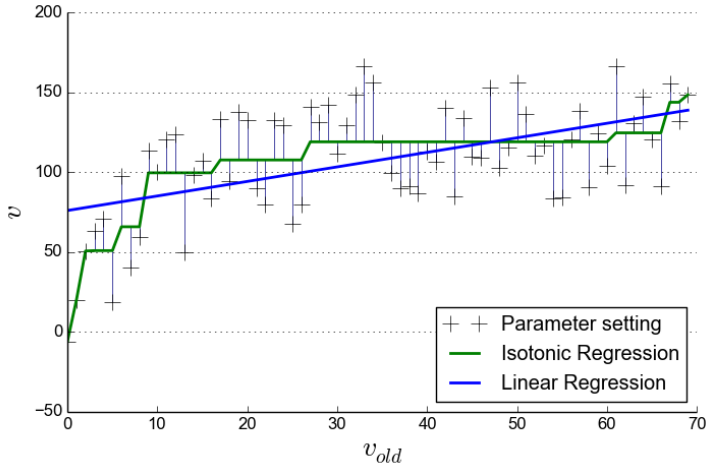


Figure 5.8: Linear and Isotonic regression [115].

Challenge (c) – How to Transfer

Knowledge is transferred differently depending on which hypothesis, if any, was not rejected. If the linear relationship hypothesis was accepted $\mathcal{H}_l$ the old values $\mathbb{y}_{old}$ or $\mathbb{v}_{old}$ are mapped to the new design. This is done by calculating a least-square linear regression to create a mapping function z_i for either f, h_i or g_i. So the mapping function z_i becomes

$$z_j(v) = av + b. \tag{5.3}$$

which is then used to perform mapping $z_j(v_{old,i}) = v_i$ and treat the data from old experiment as if it originated from the new design. The same procedure applies to vectors $\mathbb{y}_{old}$ and $\mathbb{y}$. If the linear relationship hypothesis $\mathcal{H}_l$ does not hold, but the monotonic $\mathcal{H}_m$

does, isotonic regression is performed [45]. The isotonic regression solves a weighted least squared fit problem. The problem is formulated as a quadratic program

$$\min \sum_i w_i (v_{old,i} - v_i)^2. \tag{5.4}$$

which is subject to $v_{old,min} = v_{old,1} \leq v_{old,2} \leq .. \leq v_{old,n_{\mathcal{H}}} = v_{old,max}$. The constraint enforces the monotonic relationship between the two variables. The weights w_i are strictly positive. When Spearman rank correlation is negative, the z_j function is believed to be strictly negative. In such case the problem is trivially reformulated by negating $\mathbb{v}$. The same procedure applies to vectors $\mathbb{y}_{old}$ and $\mathbb{y}$.

Table 5.4: The table presents training set created using knowledge transfer step for the designs presented in Table 5.1 and Table 5.2. Blue rows represent transferred data.

m_w	*cores*	Throughput $\mathbb{y}$	Accuracy $\mathbb{v}_1$	Latency $\mathbb{v}_2$	LUTs $\mathbb{v}_3$	Exit code $\mathbb{t}$
32	1	201.2	0.001	1 ms	25%	0
19	3	405.3	0.005	3 ms	24%	0
21	5	606.8	0.004	9 ms	57%	0
19	5	602.1^{II}	0.005^{II}	$9^{\dagger}$ ms	$\mathbf{n/a}^{+}$	$\mathbf{n/a}^{-}$
24	5	606.1^{II}	0.002^{II}	$9^{\dagger}$ ms	$\mathbf{n/a}^{+}$	$\mathbf{n/a}^{-}$
15	7	804.7^{II}	0.1^{II}	$15^{\dagger}$ ms	$\mathbf{n/a}^{+}$	2^{δ}

II Linear regression used to obtain the prediction.

† Isotonic regression used to obtain the prediction.

δ The configuration is predicted to be inaccurate based on accuracy prediction.

$+$ Could not create a regression.

$-$ Could not create LUT regression, cannot predict LUT constrain failure.

Related designs and different parameter spaces

It is possible that ranges of some of the parameters between $\mathcal{X}_{old}$ and $\mathcal{X}$ are vastly different. In that situation it can be plausible for the designer to use a mapping between them. For example, two designs are based on Monte Carlo techniques and there is a software parameter *Simulations*, which is the number of simulations used in a particular design. The range for the two designs is different for the *Simulations* parameter. One is in the range of thousands, whilst the other is in the range of millions. In such situations it is recommended to use a linear map to transform the old *Simulations* parameter to make it match that of the new design. Thus a proxy parameter is created. This is done on

case by case basis, and requires input from the designer to transform the old parameter settings to the new $\mathcal{X}$. An example is presented in the next section.

Choose a random set $\mathbb{X}_{\mathcal{H}}$ of designs from $\mathbb{X}_{old}$ for hypothesis testing;
Evaluate $\mathbb{X}_{\mathcal{H}}$ designs for the new platform using the benchmark function b;
Aggregate results for f, all h_i and g_i in corresponding vectors $\mathbb{v}_i$;
for f, all h_i and g_i **do**
    Test $\mathcal{H}_l$ using $\mathbb{v}_i$;
    if $\mathcal{H}_l$ not rejected at significance level α **then**
        Calculate linear least square regression (z_i);
        Map $\mathbb{v}_{old}$ using the function z_i;
            a.) For f, insert the results into the vector $\mathbb{y}$;
            b.) For h_i or g_i, evaluate constraints using the mapping and insert exit codes into the vector $\mathbb{t}$;
    else
        Test $\mathcal{H}_m$ using $\mathbb{v}_i$;
        if $\mathcal{H}_m$ not rejected at significance level α **then**
            Calculate Isotonic regression (z_i);
            Map $\mathbb{v}_{old}$ using the z_i function;
                a.) For f, insert the results to the vector $\mathbb{y}$;
                b.) For h_i or g_i, evaluate constraints using the mapping and insert exit codes into the vector $\mathbb{t}$;

Figure 5.9: The knowledge transfer step.

5.4 Evaluation

The primary objective of the evaluation section is to identify the benefits, and potential drawbacks, of knowledge transfer. The comparison is made between the new Auto-Transfer algorithm and ARDEGO. Three application case studies are used. A quadrature-based financial design with customizable precision [155], a high performance RTM design with seven parameters [109] and two designs based on the SMC SMCGen framework [47]. The quadrature-based financial design and the RTM design are used to identify the first use case. The designs are first optimized for one platform, followed by the optimization for a second platform. The optimization time of the design on the second platform is compared to ARDEGO. The two designs based on the SMCGen framework are used to evaluate use case two, the robot localization and stochastic volatility designs. First the robot localization design is optimized, and later those results are transferred by Auto-Transfer algorithm to optimize the stochastic volatility design.

Three potential benefits of knowledge transfer are evaluated. Firstly, lower number of initial samples allows for earlier use of the data efficient acquisition function. The configurations sampled from $\mathbb{X}_{old}$ are more likely to be valid, hence smaller number of samples can be drawn. By the data efficiency argument established in Chapter 4, earlier use of the data efficient acquisition function should help in optimization. Secondly, the sampling is biased by $\mathbb{X}_{old}$. The old design repository contains promising configurations, instead of randomly sampling from the $\mathcal{X}$ the initial samples in the Auto-Transfer algorithm are drawn from $\mathbb{X}_{old}$. Lastly, through knowledge transfer, a more accurate surrogate model can be using a smaller number of evaluations.

Auto-Transfer algorithm uses up to 5 D parameter settings to construct $\mathbb{X}_{\mathcal{H}}$. The actual number is based on the size of $\mathbb{X}_{\mathcal{H}}$, in the case of the quadrature design this usually around 2 D to 3 D with around 6 to 9 configurations. The number of initial sample parameter settings for ARDEGO is set to match that of Auto-Transfer algorithm. This evaluation method allows to assess the impact of the knowledge transfer and sampling bias, rather than the impact of lower number of initial samples. The α is set to 1%, and correlations weaker than ± 0.95 are rejected. The algorithm terminates hardware generation if, during the preliminary resource report, any of the resources exceeds the FPGA size by more than 10%. This is crucial for automated optimization, the preliminary reports gives a good indication of the final resource usage, and likely overmapping can be detected as quickly as in 20 minutes.

5.4.1 Implementation

The worker nodes consist of high performance Intel Xeon x5650 (32 nm, 6 cores, 2.67GHz) CPUs. The training data is normalized prior to model training. SVM is chosen as the classifier with an squared exponential kernel which is cross-validates on a set of parameters $\gamma \times C = \{1.2^i\}_{i=-10}^{10} \times 10\{1.25^i\}_{i=1}^{10}$. The GP is retrained 100 times with random initialization of the hyperparameters. 5000 Monte Carlo simulations are used for each $E_{MC}[I_{\mathfrak{v}}^{(\mu,\lambda)}(\mathbf{X}_\lambda)]$ estimation. [1]

[1] In Chapter 4 experiments were performed with between 5 and 500,000 Monte Carlo simulations, no noticeable impact on the algorithm performance was observed. Although experiments indicated as little as 5 simulations as sufficient, 5000 simulations were chosen as a precautionary measure while not impacting experiment time. In [75] authors suggest using adaptive simulation allocation scheme. The topic requires further study.

Table 5.5: knowledge transfer test designs overview.

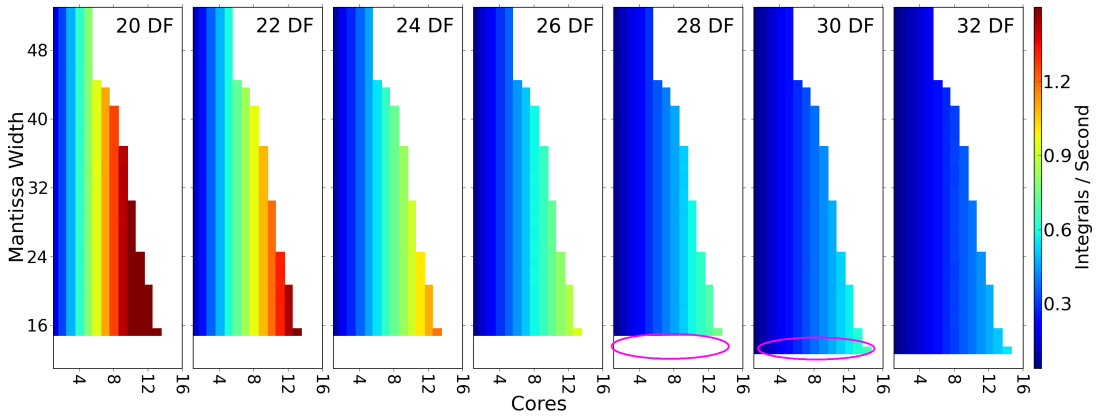
Design	**Noise**	**Constraints**	**No.**[II]	*D*
Quadrature$^{\delta\dagger}$ [155]	Performance of the design exhibits some noise. One of the parameters is a software parameter. Two benchmarks are available, a design throughput and an energy efficiency benchmarks. LUT bound	Accuracy and resource constraints.	20,000$^{\delta}$ or 10,000†	3
RTM$^{\delta\dagger}$ [109]	Not noisy. The valid portion of the design space is small.	Resource and memory bandwidth constraints.	20,1600,000$^{\delta}$ or 80,650,000†	7
Robot$^{\delta}$ [47]	Noisy benchmark function. The challenge for efficient optimization is for the algorithm to determine that one of the parameters has no impact on the design's performance.	Resource and Accuracy constraints. One of the parameters is a software parameter.	24,000	3
Stochastic$^{\delta}$ [47]	Similar to the previous design.	Resource and Accuracy constraints. One of the parameters is a software parameter.	24,000	3

II Number of possible designs in the parameter space.

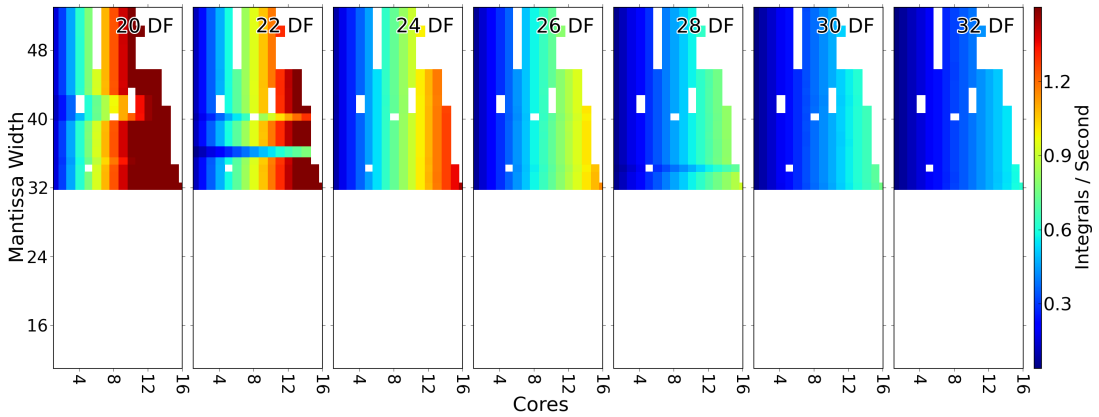
δ Optimized for Maxeler MPC-X1000 system with a Xilinx Virtex-6 XC6VSX475T FPGA.

† Optimized for Maxeler MPC-X2000 system with an Altera Stratix V GS 5SGSD8 FPGA.

ω The biggest challenge with optimization of the PQ design is PAR and timing issues.



(a) *Maxeler MPC-X1000 system with Xilinx Virtex-6 XC6VSX475T FPGA*



(b) *Maxeler MPC-X2000 Altera Stratix V GS 5SGSD8 FPGA*

Figure 5.10: Visualization of a subset of the parameter space for the throughput benchmark $\epsilon_{rms} = 0.1$ of the quadrature-based financial design implemented on two different platforms.

5.4.2 Cross-platform, Quadrature-based Financial

In [155] the designer explores tradeoff between accuracy and throughput in a quadrature-based financial design with three parameters. The design can be used to compute integrals for various financial applications. 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 estimations. All of the parameters are uniformly discrete.

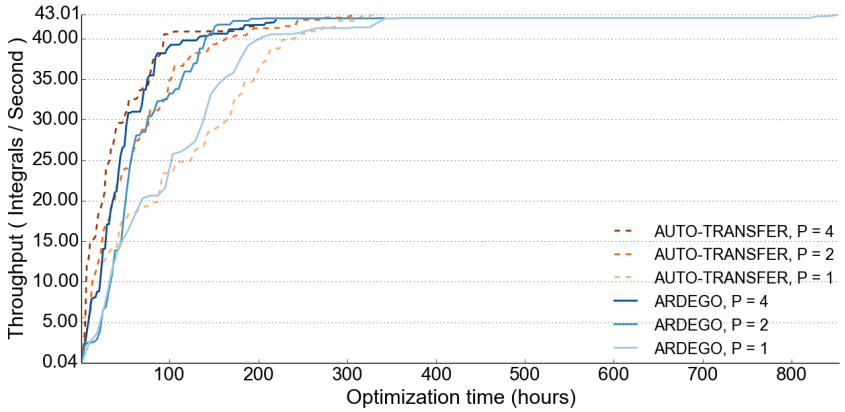


Figure 5.11: Optimization of the quadrature-based financial design throughput benchmark for $\epsilon_{rms} = 0.1$ using knowledge transfer.

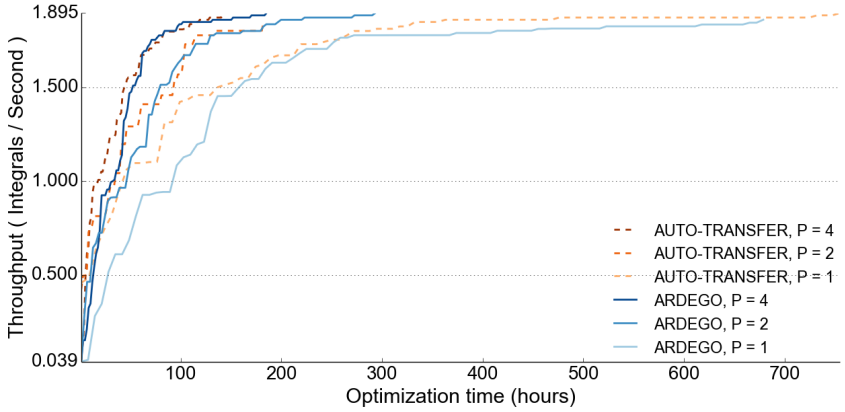


Figure 5.12: Optimization of the quadrature-based financial design throughput benchmark for $\epsilon_{rms} = 0.01$ using knowledge transfer.

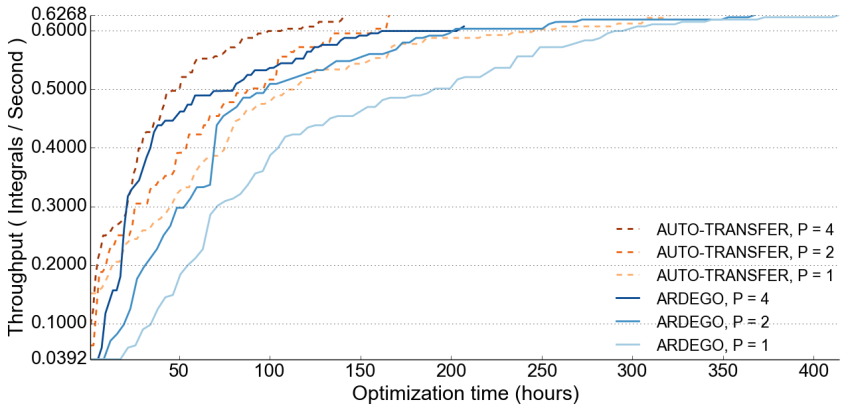


Figure 5.13: Optimization of the quadrature-based financial design throughput benchmark for $\epsilon_{rms} = 0.001$ using knowledge transfer.

The optimization goal is to find the design offering the highest throughput of integrations per second ϕ_{int} given a required minimum accuracy defined in terms of root mean square error ϵ_{rms}. The error is defined with respect to the results obtained by calculating a set of reference integrals at the highest possible precision. The ARDEGO algorithm terminates when the globally optimal configuration for a given ϵ_{rms} is found. Although some designs produce inaccurate results, the results can be transferred for regression. Resource usage is linearly related to *cores*. 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 was already generated. If a design with m_w, *cores* is evaluated, and had not been previously evaluated, a new bitstream is generated.

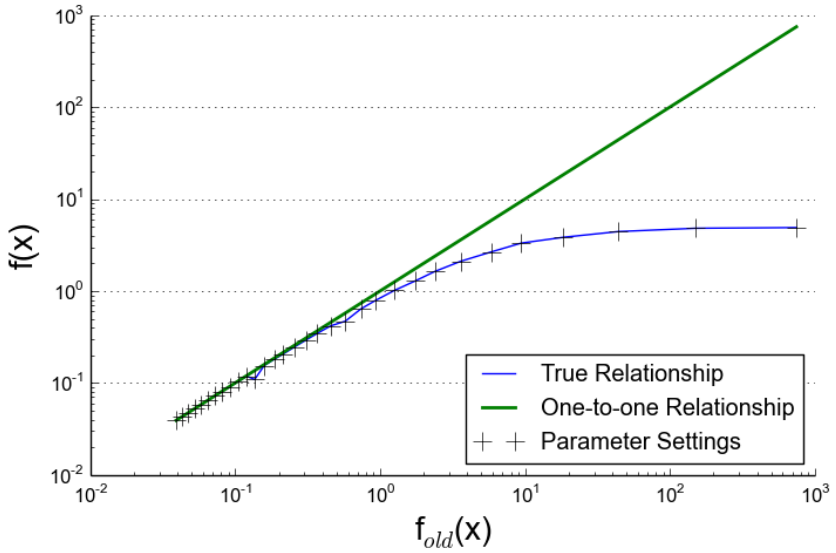


Figure 5.14: Relationship between performance of a single core design for the quadrature design implemented on the MPC-X1000 and MPC-X2000 platforms. Note the log-scale. The throughput for MPC-X2000 improves little for low d_f values, as the problem becomes communcation bound instead of compute bound. This is less of an issue in the case of MPC-X1000.

The design is first optimized for the Maxeler MPC-X1000 system with Xilinx Virtex-6 XC6VSX475T FPGA platform. Afterwards, optimization for Maxeler MPC-X2000 Altera Stratix V GS 5SGSD8 FPGA follows. Due to Altera floating point arithmetic restrictions, the parameter m_w for the new platform is restricted to the range of [32,53].

During sampling, there are parameter settings with larger d_f in the design repository $\mathbb{X}_{old}$. This means that the hypothesis tests always indicate a linear relationship, even though it is actually monotonic. For lower d_f the linear relationship holds. That is

why the speedup for $\epsilon_{rms} = 0.001$ is seen throughout all of the optimization, as seen in Figure 5.13.

Knowledge transfer speeds up optimization in nearly all of the cases during early stages of optimization as seen in Figures 5.11-5.13. This is due to knowledge transfer induced inaccuracies. Although both of the designs run at the same clock frequency, and the accuracy constraint functions of the design on both platforms are identical, this is not the case for the fitness function. Presented in Figure 5.14 is the relation between the performance of the design on the two platforms over a set of d_f. As the d_f parameter is decreased, the problem becomes communication bound instead of compute bound. This is especially prominent when $\epsilon_{rms} = 0.1$ or $\epsilon_{rms} = 0.01$.

5.4.3 Cross-platform, Reverse Time Migration

In [109] the designer faces a problem of optimizing seven parameters of a high performance RTM design; depending on the platform used, there are between 20 and 81 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 two 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 [109]. The optimized design is over 100 times faster than the basic configuration, the unoptimized basic design. All of the parameters are uniformly discrete.

The design is first optimized for the Maxeler MPC-X1000 system with Xilinx Virtex-6 XC6VSX475T FPGA platform. Afterwards optimization for Maxeler MPC-X2000 Altera Stratix V GS 5SGSD8 FPGA follows.

The RTM design is a perfect example of a design, which can benefit a lot from knowledge transfer. The parameter space for the new platform MPC-X2000 fully contains the parameter space used for building on MPC-X1000. As seen in the Figure 5.15 there is a clear benefit from using the old designs. The initial large benefit comes from reevaluation of old designs, which all build on the old platform, and from knowledge transfer. For all

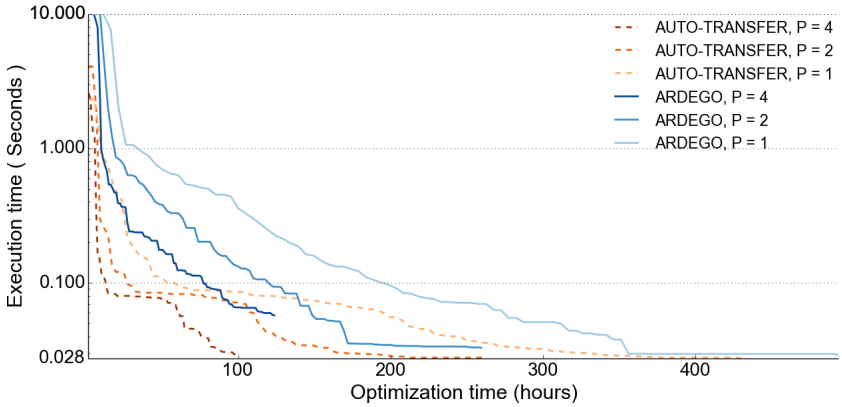


Figure 5.15: Optimization of the RTM design execution time for MPC-X2000, using knowledge resue.

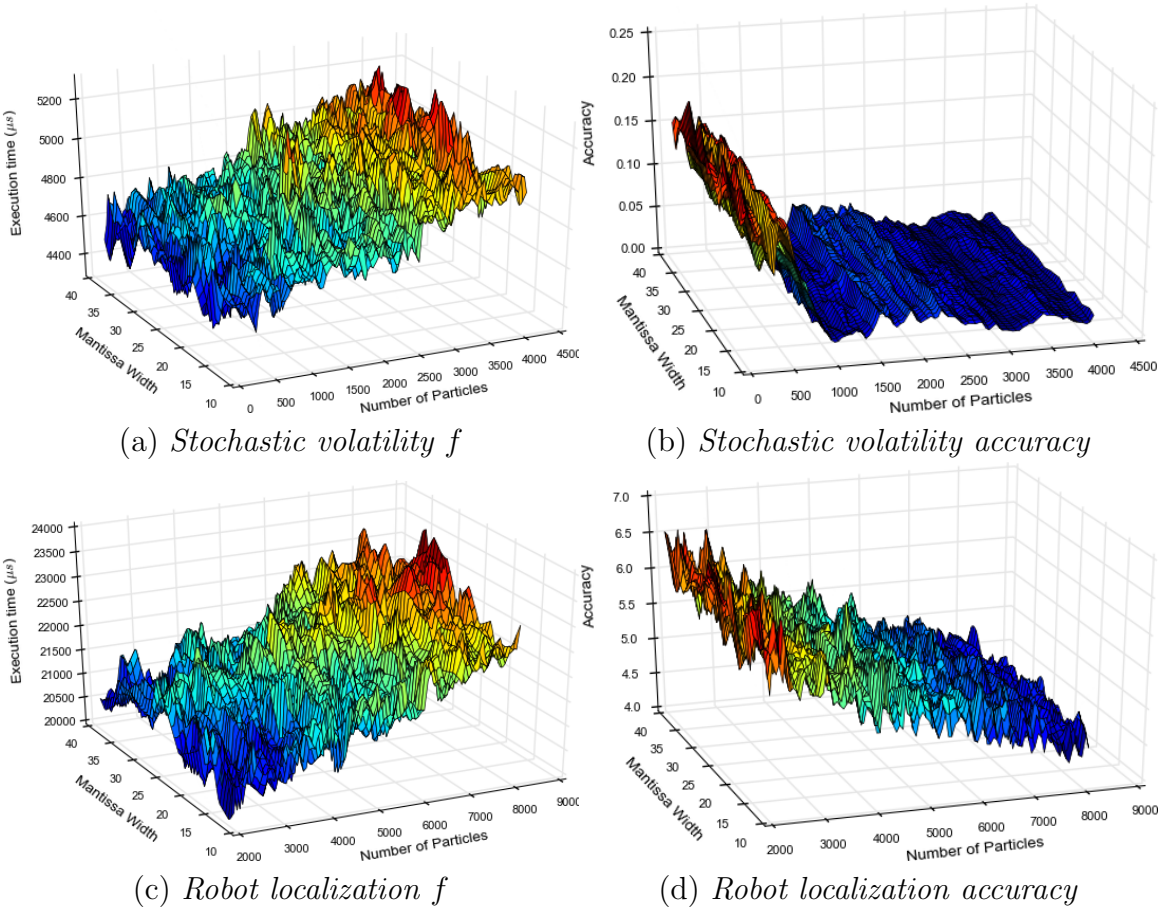
P, the algorithm ceases optimization when using knowledge for a certain amount of time. The optimization log files indicate that this is due to the algorithm refining the valid region of the parameter space, which is much larger for the new platform. As multiple invalid designs are evaluated, the optimization does not progress.

5.4.4 Related Designs, Stochastic Volatility Design and Robot Localization

The stochastic volatility and robot localization designs are implemented using the SMCGen framework [47]. The goal of SMC methods is to estimate the posterior distribution of some hidden problem states. In particular, SMC deals with problems where new observations come in a sequence, and inference has to be done on-line. SMC are simulation based methods, where a number of particles is used to model the posterior distribution.

SMC methods are applied to stochastic volatility models which are used in finance when there is no closed form solution to a problem, or the user does not wish to use approximations for pricing functions due to accuracy constraints. The volatility is then modeled as a stochastic process. SMC methods are one possible solution when the inference has to be done online. For this particular design [47], the parameters are the number of processing cores N_C, the number of particles used in the simulation N_P and mantissa width of numerical operators m_w. The number of cores N_C is limited to 64 in powers of two, the number of particles is tested on the range of 96 to 3984 in 96 increments and the range of the mantissa width m_w of the floating point operators is set from 10 to 40. The number of particles N_P is a software parameter. All of the parameters are

uniformly discrete. The chip cannot accommodate more than 16 cores. The design suffers from minor place and route as well as timing issues.



(a) *Stochastic volatility f* (b) *Stochastic volatility accuracy*

(c) *Robot localization f* (d) *Robot localization accuracy*

Figure 5.16: Stochastic volatlity and robot localization designs.

The robot localization design is implemented using the same SMC SMCGen framework [47] as the stochastic volatility design. The design is used for mobile robot localization. The robot needs to be aware of surrounding moving objects. In a sequential loop fashion the robot uses its sensors to identify its location and perform motion. The parameters are the number of processing cores N_C, the number of particles used in the simulation N_P and mantissa width of numerical operators m_w. For this particular design [47], the parameters are the number of processing cores N_C, the number of particles used in the simulation N_P and mantissa width of numerical operators m_w. The number of cores N_C is limited to 64 in powers of two, the number of particles is tested on the range of 2048 to 8096 in 96 increments and the range of the mantissa width m_w of the floating point operators is set from 10 to 40. The number of particles N_P is a software parameter. All of

the parameters are uniformly discrete. The chip cannot accommodate more than 4 cores. The design suffers from minor place and route as well as timing issues.

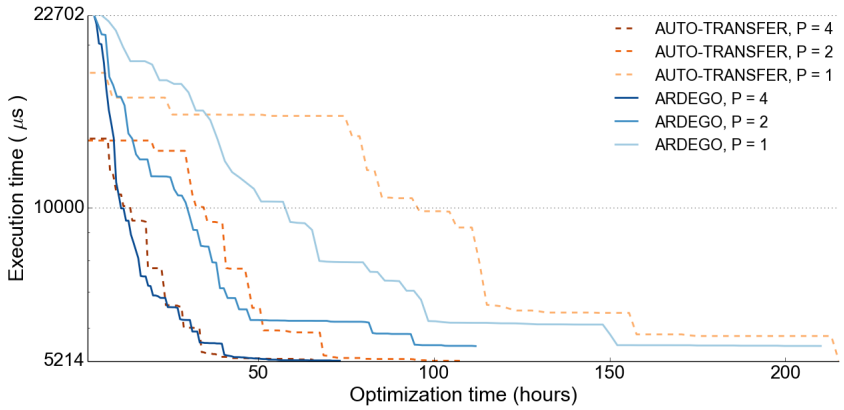


Figure 5.17: Optimization of robot localization design using knowledge transferd from the optimization of the stochastic volatility design.

For both designs the software benchmark was executed 300 times, the execution time is in the microsecond range and therefore is highly susceptible to measurement noises. The process took as much as 5-6 minutes per parameter setting. The noise is clearly visible in Figure 5.16. For both of the designs, both the accuracy and the fitness function are noisy. The challenge in optimization of those designs is for the algorithm to discover that m_w has no impact on both the accuracy and the execution time.

The knowledge transfer in those designs is tested by first optimizing the stochastic volatility design, and then using collected data to optimize the robot design. There is a clear linear relationship between fitness functions of those designs, and a monotonic relationship between their accuracy functions. The challenge is to find those relationships despite the noise and small number of available data. The designs are fairly simple, hence optimization is fast and results in little available data. The range of the N_P parameter on both platforms is normalized, so that the parameter spaces are identical. This reflects a designer assumption that the number of particles should have similar impact on both of the designs.

As seen in Figure 5.17 the benefit from knowledge transfer is none for $P = 2$ or $P = 4$, and negative when $P = 1$. The number of designs that can be built on both platforms is relatively small. The chip for the robot design does not accommodate more than 4 cores, while for the stochastic design up to 16 core designs can be built. Furthermore, this results in the algorithm incapable of discovering the monotonic relationship between the accuracy functions. With only between 3 and 5 valid parameter configuration in $\mathbb{X}_{\mathcal{H}}$, and

a noisy accuracy function, both the $\mathcal{H}_l$ and $\mathcal{H}_m$ hypothesis fail. This is a clear example of a negative impact of knowledge transfer. Although, aside of $P = 1$, the knowledge transfer has no impact on the performance of the algorithm.

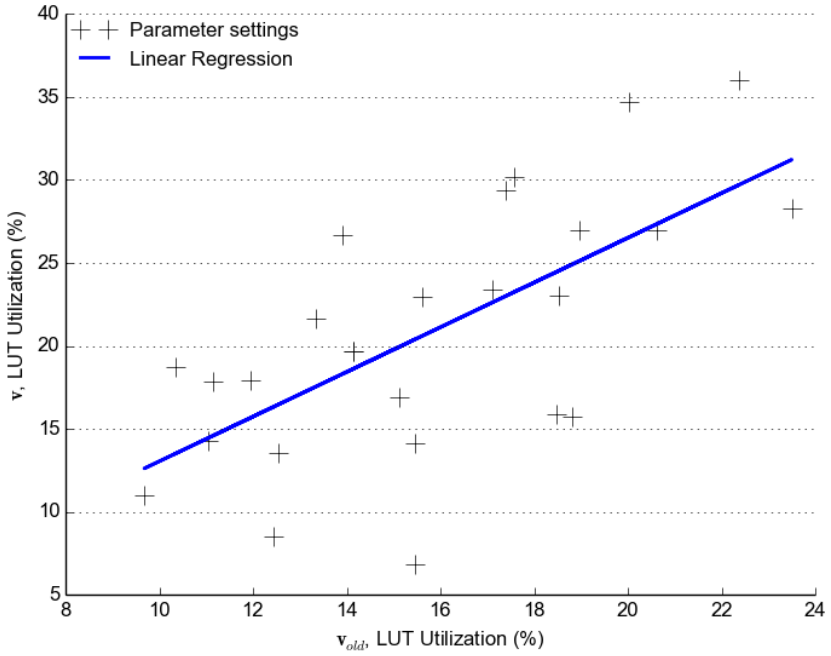


Figure 5.18: Comparison of different parameter setting LUT utilization of the robot and stochastic volatiltiy design. Stochastic volatility is the old design.

Depending on m_w both of the designs become BRAM bound, LUT bound or PAR becomes the issue. Although in the current approach it is impossible to recover information on PAR issues, the algorithm can reuse synthesis resource utilization reports. Unfortunately, it seems that in this test case they provide no positive feedback. There are a number of possible reasons for this.

Looking at Figure 5.18, there is a clear correlation between LUT utilization of the two designs. Unfortunately, it is weak, 0.66 at $\alpha = 0.0002$. This correlation was estimated using relatively large amount of data, during optimization, number of valid parameter settings in $\mathbb{X}_{\mathcal{H}}$ was typically between 3 and 5 samples. Such weak correlations, even at low α have to be treated with caution. Hence, during optimization both $\mathcal{H}_l$ and $\mathcal{H}_m$ for LUTs were rejected.

Looking at Figure 5.19, there is a a clear correlation between BRAM utilization of the two designs. Contrary to the previous case, the correlation is strong. The correlation across is 0.93 at $\alpha \approx 0.0$ significance level. The data set contains one outlier, when removed the correlation spikes to 0.987 at $\alpha \approx 0.0$, a strong correlation. Again, those

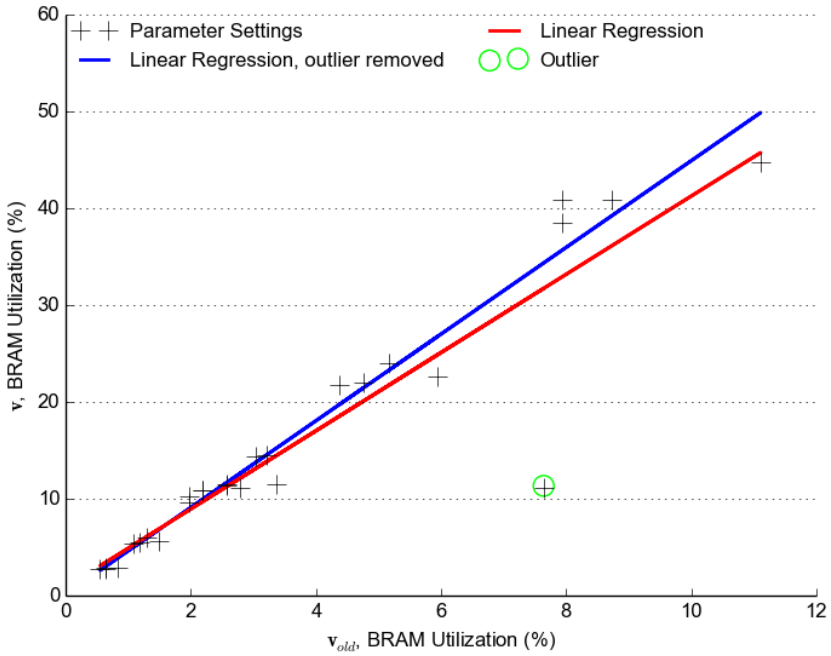


Figure 5.19: Comparison of different parameter setting BRAM utilization of the robot and stochastic volatiltiy design. Stochastic volatility is the old design.

correlations were calculated using relatively large amount of data, during experiments number of valid parameter in $\mathbb{X}_{\mathcal{H}}$ was typically between 3 and 5 samples. The $\mathcal{H}_l$ was often accepted.

5.5 Discussion

5.5.1 Results

Knowledge transfer is a challenging technique, which requires further refinement. It however shows a lot of promise.

For the quadrature design, it can offer as much as 35% reduction in optimization time. Looking at Table 5.6 when $\epsilon_{rms} = 0.001$ the reduction in optimization time when $P = 4$ is decreased from 133 hours to 86 hours, a 35% optimization time reduction. The reduction for $P = 2$ and $P = 1$ is 33% and 35% respectively. For $\epsilon_{rms} = 0.01$ a reduction is observed only for $P = 2$ and $P = 1$, 15% and 20% respectively. For $\epsilon_{rms} = 0.1$ when $P = 2$ small negative impact is observed. The impact is not as big as the optimization time in Table 5.6 suggests, the time spent on fine tuning of the design configuration to achieve the 2% improvement is substantial.

Table 5.6: The average optimization time in hours and the percentage of the average performance of the optimal configuration of different designs. ARDEGO without and with knowledge transfer is compared. The global optimum was found using exhaustive search.

Design	Quad. Th. (0.1)	Quad. Th. (0.01)	Quad. Th. (0.001)	RTM	Robot
ARDEGO $P = 1$	277 (100%)	354 (98%)	289 (100%)	311 (94%)	118 (94%)
ARDEGO $P = 1$, K.	237 (100%)	300 (100%)	**188 (100%)**$^+$	291 (100%)	134 (100%)
ARDEGO $P = 2$	129 (98%)	152 (100%)	190 (100%)	176 (86%)	63 (94%)
ARDEGO $P = 2$, K.	**183 (100%)**$^-$	121 (98%)	**127 (100%)**$^+$	**163 (99%)**$^+$	76 (100%)
ARDEGO $P = 4$	121 (98%)	86 (100%)	133 (97%)	96 (48%)	43 (100%)
ARDEGO $P = 4$, K.	117 (97%)	89 (100%)	**86 (100%)**$^+$	**85 (94%)**$^+$	39 (100%)

1 ARDEGO algorithm uses adaptive sampling plan and 5000 Monte Carlo estimates per EI evaluation.

2 The time presented is the average time when an algorithm stopped improving design's performance.

3 Numbers in parenthesis are percentages of the performance of the optimal design configuration.

+ Positive impact of knowledge transfer.

− Possibly negative impact of knowledge transfer.

For the RTM design, the optimization time reduction as well as the performance of the optimal configuration is noticeably improved by knowledge transfer. The improvement is most prominent when P is greater than 1. The improvement when $P = 4$ is 45% in design performance, as well as shorter optimization time. In the last benchmark design, knowledge transfer across related designs, no clear improvement was observed. The optimization for $P = 2$ and $P = 1$ offers better performing configurations, and for $P = 4$ the optimization time is 10% shorter. Yet, it is difficult to rule out experiment noise especially when looking at the dynamics of optimization presented in Figure 5.17.

Note that the numbers reported in Table 5.6 are less informative than the previously presented figures. Optimization plot shows the full dynamics of optimization. The algorithm terminates after its evaluation budget was exhausted, and comparison of optimization time without looking into performance of the best found design may be misleading. Lastly, there is the question of which algorithm should be preferred; The one that finds a sub-optimal, yet promising configuration in a short time span or one which takes more time to find the optimal configuration.

5.5.2 Usability

The biggest benefit from using the Auto-Transfer algorithm comes when porting a design onto a new platform. A reduction of up to two days of optimization time is observed, 133 hours to 86 hours. The algorithm needs more evaluation examples, but as no negative impact of knowledge transfer was observed. The potential substantial time saving is very promising. This is the case for both an experienced and novice user.

Knowledge transfer for related designs is not as straightforward as cross-platform transfer. It should pose no problems for an experienced designer who understands that there is a relationship between two designs, while can be problematic for a novice designer. The hypothesis testing is designed in such way to eliminate, or at least minimize, cases where relationship was either falsely indicated or can not be established. This means that with a great deal of confidence, Auto-Transfer will perform as good as ARDEGO and possibly much better.

5.6 Conclusion

We present a new Auto-Transfer algorithm which can offer substantial reduction in optimization time. It derives from work presented in Chapter 4. For the quadrature-based financial design and the reverse time migration designs we observe a reduction in optimization time of up to 35%. For the quadrature based design, the knowledge transfer step helped improve optimization speed despite large amount of noise being introduced by the new platform.

Based on our experience, and experimental work, there are a number of obvious features that can be used in knowledge transfer. Specific to reconfigurable computing is the resource utilization; that includes DSPs, BRAMs, and LUTs utilization. Design specific features like accuracy, throughput, latency or power are good candidates. Using detailed synthesis timing reports to better optimize clock frequency is more challenging, propagation delay information also includes circuit spatial information and could be misleading.

Whenever the parameter spaces of the two designs have a large common set and enough configurations available to verify their relationship, knowledge transfer can be expected to be beneficial. This generally will not be a concern when porting designs across platforms or slightly revising their source code. For related designs, the designer has to be careful. Knowledge transfer is not always beneficial, and it cannot guarantee to improve algorithm's performance. Minor negative impact, in the form of less desirable optimization dynamics, was observed in one of the test cases. Currently the robustness and stability of the approach require further study.

Chapter 6

Conclusion and Future Work

In this thesis we show the potential for automatic optimization of reconfigurable designs. Furthermore, we extend the idea to show how results of synthesis and benchmarking can be used to speedup optimization. In this chapter, we conclude the achievements and suggest future work to address the current limitations.

6.1 Summary of Achievements

The main focus of the thesis is decreasing the effort associated with reconfigurable design development by automating optimization. The designer should focus on the problem statement and designs correctness, the tools should carry out implementation. Currently, the design cycle has limited automation. This is being partially addressed by HLS to describe the design [49]. HLS is important in the context of reconfigurable computing as it largely makes the underlying platform transparent to the programmer. Through the use of libraries and integrated tool-chains hardware compilers provide an programming model API. Good examples are ROCCC 2.0 [18], YAHDL [39] or its evolution the MaxCompiler [3]. Yet, even when using HLS user faces a number of different design choices. In particular, this thesis focused on optimization of design parameters. In the background section three major challenges were identified, for readers convenience the table is again provided in Table 6.1.

Our first step towards automation is presented in **Chapter 3**. This work presents initial automation attempt, addressing long hardware generation time mentioned in Table 6.1. The MLO algorithm is based on surrogate model technology and particle swarm optimization. The method shows a lot of promise, we show how it can optimize designs up to 50% faster then design specific optimization methodology. The optimization can

Table 6.1: The three major challenges with optimization of reconfigurable designs.

Challenge	Description
Hardware Generation Time	Generation and evaluation of a single parameter setting takes between an hour to two days.
No. of possible configurations	The curse of dimensionality. The number of potential design configurations can be in the millions.
Lost Knowledge	The knowledge gathered during development and optimization of designs is not transferred.

involve both software and hardware parameters. Yet it suffers from a number of problems; complicated optimization loop and lack of parallelism. Reconfigurable design automatic optimization involves multiple hardware generations, a time consuming process. The optimization time is generally in 50-150 hours for three parameter designs. Lastly, the approach fails to optimize designs with much large number of parameters.

The complicated optimization loop, lack of parallelism and inability to deal with high dimensional problems are addressed in **Chapter 4**. Based on experience gathered during MLO evaluation, a new algorithm is presented, called ARDEGO. The algorithm addresses the first two challenges mentioned in Table 6.1. The new algorithm is based on the EGO algorithm and it offers several improvements over MLO. It offers parallelism, and has a straight-forward optimization loop. The optimization time is decreased by as much as 50% for a single optimization node. As the ARDEGO algorithm uses multiple optimization workers, its optimization speed is even further increased relative to MLO. When using 6 worker nodes, optimization finishes in as little as one fifth of the time. The ARDEGO algorithm is shown to optimize a 7 parameter design in less than 93 hours, as well as reduce the optimization time of the quadrature-based design from 32 to 25 hours compared to design specific tool when using multiple workers, a 22% reduction. The parallelization is asynchronous, taking into account big differences in hardware generation time for different parameter settings. Lastly, the Monte Carlo compute intensive component of the ARDEGO algorithm is accelerated, achieving up to a 43x speed-up.

The main drawback of ARDEGO compared to a human designers is that the algorithm does not build up its knowledge with subsequent design optimization. This is the last of the challenges mentioned in Table 6.1. **Chapter 5** shows how results of design synthesis and design benchmarking can be transferred to decrease optimization time of a similar design when ported onto a different platform or when its code is revised. In modern dynamic

environment the designs are often modified and the underlying platforms revised. A methodology to transfer old synthesis results is crucial to automation. The initial findings show clear benefit when applying knowledge transfer when porting a design onto a different platform, up to a 35% optimization time reduction is observed. Knowledge transfer does not always offer improved optimization time or better performing configurations, the concept requires extra case studies and further work.

6.2 Future Work

Although the work presented in this thesis shows a lot of promise, a number of open questions arise.

6.2.1 Revision of ARDEGO

ARDEGO algorithm was a natural evolution of the MLO algorithm. Optimization was shifted from metaheursitics to Bayesian optimization and the algorithm was parallelized. The current main limitations of the algorithm come from two different aspects.

Evaluation Cost

First, the cost of design generation is not taken into account during optimization. This is especially important when hardware and software are co-optimized. Exhaustive evaluation of all software parameters using a given bitstream is wasteful, at the same time treating them equally to hardware parameters increases the problem complexity. One promising solution would be to split design evaluation time $\mathfrak{C}(\mathbf{x})$ into hardware generation $\mathfrak{C}_h(\mathbf{x})$ and software $\mathfrak{C}_s(\mathbf{x})$. Those two functions could be then modeled for example using GP or some other regression techniques. The incorporation of cost can be easily done by modification of the acquisition function, as shown in [107, 139, 64, 148]. In case of standard EI the cost-aware improvement function in the reconfigurable computing context [107] becomes

$$I(\mathbf{x})_c = \frac{I(\mathbf{x})}{\mathfrak{C}_s(\mathbf{x}_*) + \mathfrak{C}_h(\mathbf{x}_*)}. \tag{6.1}$$

Although this approach seems promising, it suffers from one main problem. Same as pure EGO algorithm, it is still a greedy approach. Whenever a bitstream is generated, it drastically decreases the evaluation cost of all the parameter settings which use it. At

every step of the cost-aware EGO algorithm improvement per cost will look promising, and the algorithm can end up evaluating a wasteful amount of parameter settings which offer minuscule improvement. For example, there are two alternative choices, one is a parameter setting $\mathbf{x}$ which offers improvement of a 100 at a cost of 10 hours and there is a set of one million parameter configurations which offer improvement of 1 at a cost of 1 second. Although selection of one of the cheap and less promising configurations will result in a revised model, they can still look deceitfully promising. Multi-step acquistion function would seem to be a solution, where looks few steps ahead and makes more strategic choices. But, as is suggested in [64], it is intractable [148]. Another possible mitigation of the problem is to balance the cost and improvement, using a similar concept to standard deviation and mean balancing used in the GP-UCB algorithm [26]. In the context of global optimization the acquisition function could be defined as

$$\underset{\mathbf{x}_* \in \mathcal{X}}{\operatorname{argmax}} [\bar{f}(\mathbf{x}_*) + \alpha_t^{1/2} \sigma(\mathbf{x}_*) + \frac{\gamma_t}{\mathfrak{C}_s(\mathbf{x}_*) + \mathfrak{C}_h(\mathbf{x}_*)}]. \tag{6.2}$$

where α_t and γ_t are domain-specific time-varying parameters, which regulate the trade-off between exploration through maximization of predictive mean, standard deviation and the cost of configuration evaluation. The main challenge is management of γ_t and asynchronous parallelization of such algorithm. Work on batched GP-UCB algorithm [54] could be extended.

The concept of configuration cost management is similar to typically used by human designer. For example, optimize a single core design first. Single core design is smaller and takes much less time to generate hardware. Once the single core design was optimized, increase the number of cores.

Classification and Constraint Function Modeling

Currently synthesis results, like resource utilization, are not directly used to predict the probability $\mathfrak{p}_t(\mathbf{x}_*)$ that an unseen configuration will meet all constraints. Furthermore, some of the constraints, like resource utilization, could be analytically modeled for each target device [117]. As the modeling is done per device instead of design, the approach is still generic.

Secondly, the SVM classifier does not take into account the random nature of certain aspects of hardware generation. This becomes especially apparent when looking at the PQ design, presented in Figure 6.2. There is a certain probability $\mathfrak{p}_1(\mathbf{x}_*)$ that a design

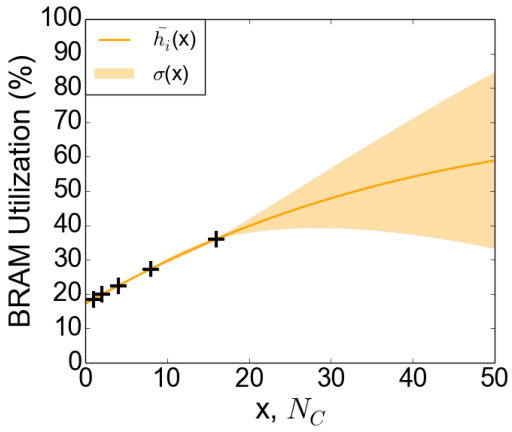


Figure 6.1: GP prediction of BRAM utilization for the stochatic volatility design.

will meet all of the constraints. The probability is related to the parameter settings.

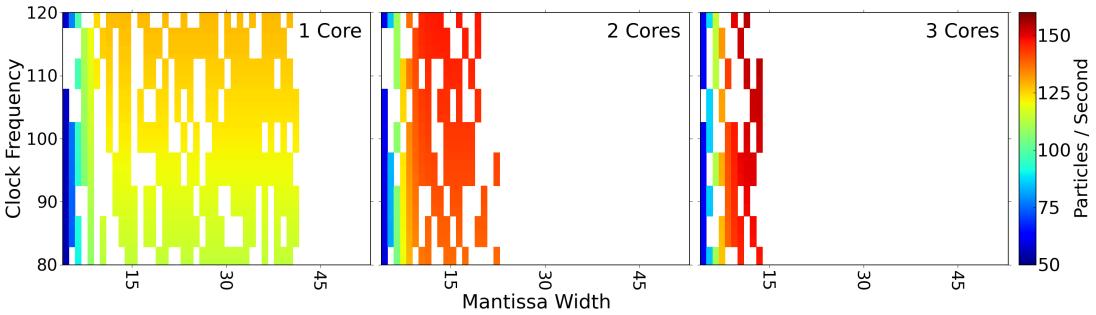
The approach requires different approach to dealing with the random constraints, a possible solution lies in the approach presented in [64]. The surrogate model should consist of a number of constraint function regressors. To make the approach truly Bayesian, SVM classification could be replaced by modeling of the constraint function using GPs, that is $h_i(\mathbf{x}) \sim \mathcal{GP}(m_i(\mathbf{x}), k_i(\mathbf{x}, \mathbf{x}))$ and $g_j(\mathbf{x}) \sim \mathcal{GP}(m_j(\mathbf{x}), k_j(\mathbf{x}, \mathbf{x}))$ each with a distinct mean and kernel function. Example of a GP fit to BRAM utilization data for stochastic volatility design with $m_w = 38$ is presented in Figure 6.1. [1] Having that prediction, one can either use the mean prediction $\bar{h}_i(\mathbf{x}_*) \geq \mathfrak{c}_i$ to construct a hard decision function, or use use the fact that GP returns a point distribution $h_i(\mathbf{x}_*) \sim \mathcal{N}(\bar{h}_i(\mathbf{x}_*), \sigma_i(\mathbf{x}_*))$. Lastly, the GP regressions could be further refined, using kernels most appropriate to different constraint functions.

Two crucial to reconfigurable computing functions, the timing function and the place and route function, should be to discover the probability a that the hardware will be generated. Those functions are vastly different to other constraint functions, like resource constraint function, which can be considered to be largely deterministic. One of the possible solutions is to use Bayesian classifier like RVM or again Gaussian Process classification.

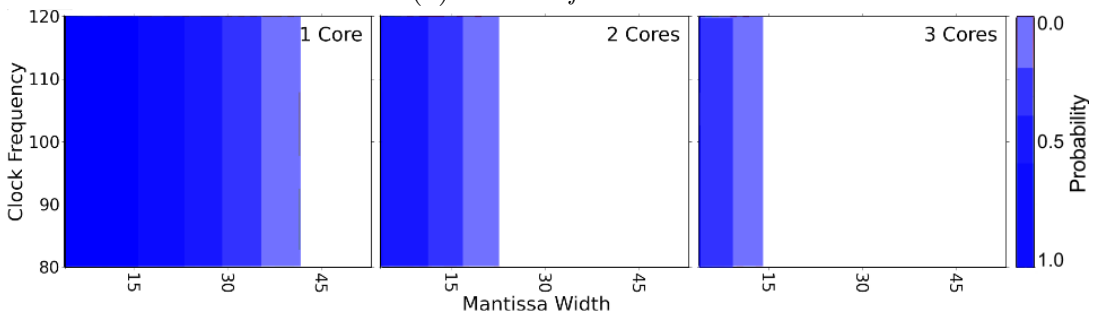
Multiobjective Optimization

Reconfigurable designs are often optimized for multiple goals, mainly performance and power. Initial investigation of modification of MLO to accommodate multiobjective

[1] More accurately, as the BRAM constraint function is only positive, $h_{bram} : \mathcal{X} \to \mathcal{R}^+$, modeling in the log space would be more appropriate $\log(h_{bram}(\mathbf{x})) \sim \mathcal{GP}(m_{bram}(\mathbf{x}), k_{bram}(\mathbf{x}, \mathbf{x}))$



(a) *Fitness function*



(b) *Probability of hardware generation*

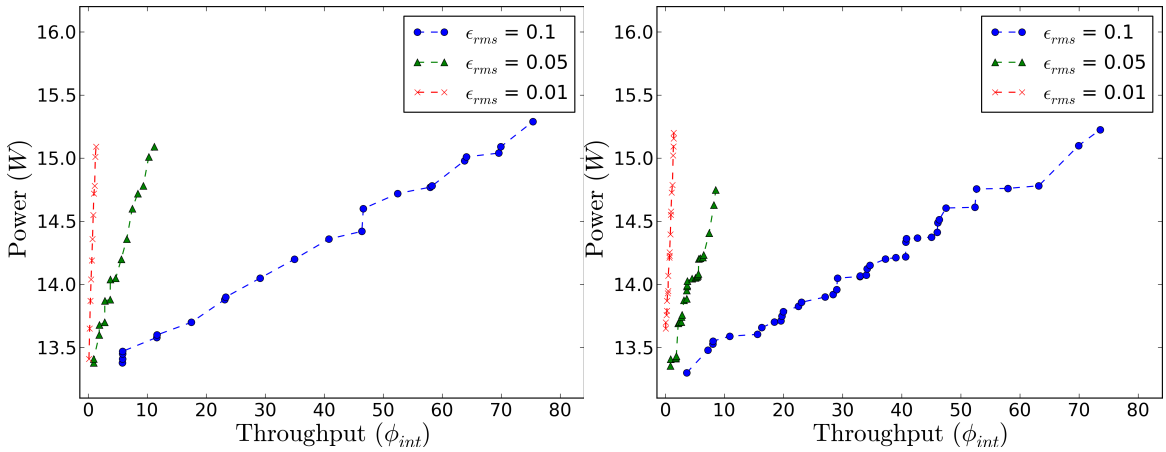
Figure 6.2: PQ design throughput fitness function [46] and probability of succesful hardware generation visualization.

optimization is presented in [90]. An example of Pareto fronts found using Multiobjective Machine Learning Optimizer (MOMLO) algorithm are presented in Figure 6.3. The goal in such optimization is to identify the Pareto optimal front for the multiple goals. As we presented in Chapter 4, ARDEGO has several benefits over surrogate model assisted meta-heuristic optimization. It uses a simpler optimization loop and according to our study offers better performance. A natural step would be to use a multiobjective Bayesian optimization algorithm [84] in the reconfigurable design optimization context.

6.2.2 Knowledge Transfer

Knowledge transfer shows promise, yet the three earlier identified tasks need to be further addressed: (a) "what to transfer", (b) "when to transfer" and (b) "how to transfer" [113].

"What to transfer"; there are a number of sources that can potentially benefit new optimization. In Chapter 5 we exploit information collected during generation of a design for a different platform. Platform specific information can be aggregated as well, possibly to allow better assessment of PAR and timing issues for different groups of related designs. Furthermore, we only investigated reusing information from a single old design. The



(a) *Real Pareto front* (b) *Estimated Pareto front*

Figure 6.3: Real and approximated Pareto fronts found with the MOMLO algorithm [90].

knowledge transfer concept can be extended to benefit from information gathered during optimization of multiple designs.

"When to transfer"; one of the main problems is a rigorous definition of related designs. The crucial work is to establish a reconfigurable design-specific distance metric that would allow for easy identification of related designs. A similar concept is presented in [116]. They define the concept of "a problem specific distance metric over decision variables that correlates with the strength of interactions between the variables". One of the possibilities is to define designs as graphs, and use graph similarity [164]. It is a compute intensive process; however, it can might allow for search for related designs across design databas. Then, multiple similar designs could be automatically fetched. Although far fetched, a cloud base multi-user design database might be the future of reconfigurable computing.

"How to transfer"; the information learned during previous optimization, for example captured in the form of optimized hyperparameters, is lost. By revising the algorithm, as described in the previous Section to accommodate new fully Bayesian surrogate model, the Knowledge Transfer approach could be substituted by a full transfer learning approach. The suggested changes were presented in the previous section, and follow similar concepts to the ones presented in [64]. Currently, a full-fledged transfer learning approach is impossible due to the lack of common design features. Construction and evaluation of such a feature set would only be possible after examination of a large data set of optimized designs. Currently such a data set is not available and is prohibitively expensive to construct, although this might change due to cloud computing and cheaper computing resources.

Bibliography

[1] http://http://www.cwcdefense.com. Retrieved July 20, 2015.

[2] http://www.nallatech.com. Retrieved July 20, 2015.

[3] http://www.maxeler.com. Retrieved July 20, 2015.

[4] https://www.khronos.org/opencl. Retrieved July 20, 2015.

[5] https://cleditor.codeplex.com. Retrieved July 20, 2015.

[6] https://www.altera.com. Retrieved July 20, 2015.

[7] http://www.xilinx.com. Retrieved July 20, 2015.

[8] http://www.intel.com. Retrieved July 20, 2015.

[9] http://www.samsung.com. Retrieved July 20, 2015.

[10] http://www.eecg.toronto.edu/~vaughn/challenge/fpga_arch.html. Retrieved July 20, 2015.

[11] https://www.java.com. Retrieved July 20, 2015.

[12] https://web.archive.org/web/20070412183416/http://filebox.vt.edu/users/tmagin/history.htm. Retrieved July 20, 2015.

[13] http://www.edn.com/electronics-news/4320763/No-room-for-Second-Place. Retrieved July 20, 2015.

[14] http://www.theplatform.net/2015/07/29/why-hyperscalers-and-clouds-are-pushing-in Retrieved July 20, 2015.

[15] http://www.xilinx.com/support/documentation/sw_manuals/xilinx12_1/mb_ref_guide.pdf. Retrieved July 20, 2015.

[16] `http://www.xilinx.com/tools/coregen.htm`. Retrieved July 20, 2015.

[17] `https://www.altera.com/content/dam/altera-www/global/en_US/pdfs/literature/ug/ug_intro_to_megafunctions.pdf`. Retrieved July 20, 2015.

[18] `http://www.jacquardcomputing.com/roccc`. Retrieved July 20, 2015.

[19] `http://www-ai.cs.uni-dortmund.de/weblab/static/api_docs/pyGPs`. Retrieved July 20, 2015.

[20] M. Abramowitz and I. Stegun. *Handbook of Mathematical Functions*. Dover Publications, 1965.

[21] C. S. Adjiman, I. P. Androulakis, and C. A. Floudas. Global optimization of MINLP problems in process synthesis and design. *Computers & Chemical Engineering*, 21:445–450, 1997.

[22] C. S. Adjiman, I. P. Androulakis, and C. A. Floudas. Global optimization of mixed-integer nonlinear problems. *AIChE Journal*, 46:176–9, Sep 2000.

[23] G. Afonso, Z. Baklouti, D. Duvivier, R. Ben Atitallah, E. Billauer, and S. Stilkerich. Heterogeneous CPU/FPGA Reconfigurable Computing System for Avionic Test Application. In *Parallel and Distributed Processing Symposium Workshops PhD Forum*, pages 260–267, May 2013.

[24] J. Arram, W. Luk, and P. Jiang. Ramethy: Reconfigurable acceleration of bisulfite sequence alignment. In *ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, pages 250–259, 2015.

[25] C. Audet, A. J. Booker, J. Dennis Jr, P. D. Frank, and D. W. Moore. A surrogate-model-based method for constrained optimization. *AIAA Journal*, 4891, 2000.

[26] P. Auer. Using confidence bounds for exploitation-exploration trade-offs. *Journal of Machine Learning Research*, 3:397–422, Mar 2003.

[27] Z. Baklouti, D. Duvivier, R. Ben Atitallah, A. Artiba, and N. Belanger. Proceedings of International Conference on Real-time simulator supporting heterogeneous CPU/FPGA architecture. In *Industrial Engineering and Systems Management*, pages 1–8, Oct 2013.

[28] A. Basudhar, C. Dribusch, S. Lacaze, and S. Missoum. Constrained efficient global optimization with support vector machines. *Structural and Multidisciplinary Optimization*, 46(2):201–221, Jan 2012.

[29] M. S. Bazaraa. *Nonlinear Programming: Theory and Algorithms*. Wiley Publishing, May 2013.

[30] T. Becker, W. Luk, and P. Y. Cheung. Parametric design for reconfigurable software-defined radio. In *Reconfigurable Computing: Architectures, Tools and Applications*, volume 5453 of *Lecture Notes in Computer Science*, pages 15–26. 2009.

[31] A. Ben-Hur and J. Weston. A user's guide to support vector machines. In *Data Mining Techniques for the Life Sciences*, volume 609, pages 223–239. Humana Press, Oct 2010.

[32] J. Bergstra and Y. Bengio. Random search for hyper-parameter optimization. *Journal of Machine Learning Research*, 13:281–305, Mar 2012.

[33] D. Berry and B. Fristedt. *Bandit Problems: Sequential Allocation of Experiments*. Chapman & Hall, 1985.

[34] M. Bhattacharya. Expensive optimisation: A metaheuristics perspective. *CoRR*, abs/1303.2215, 2013.

[35] C. M. Bishop. *Pattern Recognition and Machine Learning*. Springer-Verlag, Secaucus, NJ, USA, 2006.

[36] J. Bock and J. Hettenhausen. Discrete particle swarm optimisation for ontology alignment. *Information Sciences*, 192:152–173, Jun 2012.

[37] D. Bonett and T. Wright. Sample size requirements for estimating pearson, kendall and spearman correlations. *Psychometrika*, 65(1):23–28, 2000.

[38] B. E. Boser, I. M. Guyon, and V. N. Vapnik. A training algorithm for optimal margin classifiers. In *Proceedings of Annual Workshop on Computational Learning Theory*, pages 144–152, 1992.

[39] J. A. Bower, W. N. Cho, and W. Luk. Unifying FPGA hardware development. In *Proceedings of International Conference on Field-Programmable Technology*, pages 113–120, Jan 2007.

[40] S. Boyd, S.-J. Kim, L. Vandenberghe, and A. Hassibi. A tutorial on geometric programming. *Optimization and Engineering*, 8(1):67–127, 2007.

[41] S. Bradley, A. Hax, and T. Magnanti. *Applied Mathematical Programming*. Addison-Wesley Publishing Company, 1977.

[42] E. Brochu, T. Brochu, and N. de Freitas. A bayesian interactive optimization approach to procedural animation design. In *SIGGRAPH/Eurographics Symposium on Computer Animation*, pages 103–112, Aire-la-Ville, Switzerland, Switzerland, 2010. Eurographics Association.

[43] C. Brugger, J. A. Varela, N. Wehn, S. Tang, and R. Korn. Reverse Longstaff-schwartz American Option Pricing on Hybrid CPU/FPGA Systems. In *Proceedings of Design, Automation & Test in Europe Conference & Exhibition*, pages 1599–1602, 2015.

[44] D. Buche, N. Schraudolph, and P. Koumoutsakos. Accelerating evolutionary algorithms with Gaussian process fitness function models. *IEEE Transactions on Systems, Man, and Cybernetics*, 35(2):183–194, May 2005.

[45] N. Chakravarti. Isotonic median regression: A linear programming approach. *Mathematics of Operations Research*, 14(2):303–308, May 1989.

[46] T. Chau, K.-W. Kwok, G. Chow, K. H. Tsoi, K.-H. Lee, Z. Tse, P. Cheung, and W. Luk. Acceleration of real-time proximity query for dynamic active constraints. In *Proceedings of International Conference on Field-Programmable Technology*, pages 206–213, Dec 2013.

[47] T. C. Chau, M. Kurek, J. S. Targett, J. Humphrey, G. Skouroupathis, A. Eele, J. Maciejowski, B. Cope, K. Cobden, P. Leong, P. Y. Cheung, and W. Luk. Smc-gen: Generating reconfigurable design for sequential monte carlo applications. In *Proceedings of International Symposium on Field-Programmable Custom Computing Machines*, pages 141–148, May 2014.

[48] C. B. Cho, W. Zhang, and T. Li. Modeling and analyzing the effect of microarchitecture design parameters on microprocessor soft error vulnerability. In *Modeling, Analysis and Simulation of Computers and Telecommunication Systems*, pages 1–10, Sep 2008.

[49] J. Cong, B. Liu, S. Neuendorffer, J. Noguera, K. Vissers, and Z. Zhang. High-level synthesis for FPGAs: From prototyping to deployment. *Computer-Aided Design of Integrated Circuits and Systems*, 30(4):473–491, April 2011.

[50] C. Cortes and V. Vapnik. Support-vector networks. *Machine Learning*, 20(3):273–297, Sep 1995.

[51] B. Dammak, R. Benmansour, S. Niar, M. Baklouti, and M. Abid. A mixed integer linear programming approach for design space exploration in fpga-based mpsoc. In *Proceedings of International Conference on Field-Programmable Technology*, pages 1–4, Sep 2014.

[52] F.-M. De Rainville, F.-A. Fortin, M.-A. Gardner, M. Parizeau, and C. Gagné. DEAP: A Python Framework for Evolutionary Algorithms. In *Proceedings of Annual Conference Companion on Genetic and Evolutionary Computation*, pages 85–92. ACM, 2012.

[53] M. del Mar Hershenson, S. P. Boyd, and T. H. Lee. Optimal design of a cmos op-amp via geometric programming. *IEEE Transactions on Computer-Aided Design*, 20:1–21, 2001.

[54] T. Desautels, A. Krause, and J. W. Burdick. Parallelizing exploration-exploitation tradeoffs in Gaussian process bandit optimization. *Journal of Machine Learning Research*, 15:3873–3923, 2014.

[55] J.-P. Deschamps, G. D. Sutter, and E. Cant. *Guide to FPGA Implementation of Arithmetic Functions*. Springer Publishing Company, 2014.

[56] J. Djolonga, A. Krause, and V. Cevher. High-dimensional Gaussian process bandits. In *Proceedings of Conference on Neural Information Processing Systems*, volume 26, pages 1025–1033. Curran Associates, Inc., Dec 2013.

[57] C. Dubach. Using machine-learning to efficiently explore the architecture/compiler co-design space. The University of Edinburgh, 2009.

[58] R. Eberhart and Y. Shi. Comparing inertia weights and constriction factors in particle swarm optimization. In *Congress on Evolutionary Computation*, volume 1, pages 84–88, 2000.

[59] B. Eizad, A. Doshi, and A. Postula. FPGA based stability system for a small-scale quadrotor unmanned aerial vehicle. In *Proceedings of FPGAWorld Conference*, pages 1–6, 2011.

[60] B. Eric, N. D. Freitas, and A. Ghosh. Active preference learning with discrete choice data. In *Proceedings of Conference on Neural Information Processing Systems*, volume 20, pages 409–416. 2007.

[61] G. Estrin. Organization of computer systems: The fixed plus variable structure computer. In *Western Joint IRE-AIEE-ACM Computer Conference*, pages 33–40, May 1960.

[62] A. Forrester, A. Sobester, and A. Keane. *Engineering Design Via Surrogate Modelling: A Practical Guide*. Progress in astronautics and aeronautics. Wiley, 2008.

[63] H. Gao, W. Xu, J. Sun, and Y. Tang. Multilevel thresholding for image segmentation through an improved quantum-behaved particle swarm algorithm. *IEEE Transactions on Instrumentation and Measurement*, 59(4):934–946, Apr 2010.

[64] M. A. Gelbart, J. Snoek, and R. P. Adams. Bayesian optimization with unknown constraints. In *CoRR*, Mar 2014.

[65] D. Ginsbourger, R. Le Riche, and L. Carraro. A Multi-points Criterion for Deterministic Parallel Global Optimization based on Gaussian Processes. Technical report, HAL technical report no. hal-00260579, Mar 2008.

[66] S. Guoshao and J. Quan. A cooperative optimization algorithm based on Gaussian process and particle swarm optimization for optimizing expensive problems. In *CSO*, volume 2, pages 929–933, 2009.

[67] P. Hajela and J. Lee. Genetic algorithms in multidisciplinary rotor blade design. In *Conference on Structures, Structural Dynamics, and Material*, 1998.

[68] N. Hansen. The CMA Evolution Strategy: A comparing review. In *Towards a New Evolutionary Computation*, volume 192, pages 75–102. Jun 2006.

[69] D. Haussler and M. Opper. Mutual information, metric entropy and cumulative relative entropy risk. *Ann. Statist.*, 25(6):2451–2492, 12 1997.

[70] J. M. Hernández-Lobato, M. W. Hoffman, and Z. Ghahramani. Predictive entropy search for efficient global optimization of black-box functions. In *Proceedings of Conference on Neural Information Processing Systems*, volume 27, pages 918–926. Curran Associates, Inc., Dec 2014.

[71] B. Holland, A. D. George, H. Lam, and M. C. Smith. An analytical model for multilevel performance prediction of multi-FPGA systems. *ACM Transactions on Reconfigurable Technology and Systems*, 4(3):1–28, Aug 2011.

[72] X. Hu and R. Eberhart. Solving constrained nonlinear optimization problems with particle swarm optimization. In *World Multiconference on Systemics, Cybernetics and Informatics, year = 2002, pages = 203-206.*

[73] B. il Koh, A. D. George, R. T. Haftka, and B. J. Fregly. Parallel asynchronous particle swarm optimization. In *INTERNATIONAL JOURNAL FOR NUMERICAL METHODS IN ENGINEERING*, volume 67, pages 578–595, 2006.

[74] J. Janusevskis, R. Le Riche, and D. Ginsbourger. Parallel expected improvements for global optimization: summary, bounds and speed-up. Technical report, HAL id:hal-00613971, Aug 2011.

[75] J. Janusevskis, R. Le Riche, D. Ginsbourger, and R. Girdziusas. Expected improvements for the asynchronous parallel global optimization of expensive functions: Potentials and challenges. In *International Conference on Learning and Intelligent Optimization*, pages 413–418, Jan 2012.

[76] Q. Jin, T. Becker, W. Luk, and D. Thomas. Optimising explicit finite difference option pricing for dynamic constant reconfiguration. In *Proceedings of International Symposium on Field-Programmable Custom Computing Machines*, pages 165–172, Aug 2012.

[77] Q. Jin, W. Luk, and D. Thomas. On Comparing Financial Option Price Solvers on FPGA. In *Proceedings of International Symposium on Field-Programmable Custom Computing Machines*, pages 89–92, May 2011.

[78] Y. Jin, M. Olhofer, and B. Sendhoff. A framework for evolutionary optimization with approximate fitness functions. *IEEE Transactions on Evolutionary Computation*, 6:481–494, Dec 2002.

[79] D. R. Jones, M. Schonlau, and W. J. Welch. Efficient global optimization of expensive black-box functions. *Journal of Global Optimization*, 13(4):455–492, Dec 1998.

[80] N. Kapre, B. Chandrashekaran, H. Ng, and K. Teo. Driving timing convergence of FPGA designs through machine learning and cloud computing. In *Proceedings of International Symposium on Field-Programmable Custom Computing Machines*, May 2015.

[81] M. N. Katehakis and J. Veinott, Arthur F. The multi-armed bandit problem: Decomposition and computation. *Journal of Mathematics of Operations Research*, 12(2):262–268, May 1987.

[82] S. Kilts. *Advanced FPGA Design: Architecture, Implementation, and Optimization.* Wiley, 2007.

[83] R. Kirchgessner, G. Stitt, A. George, and H. Lam. VirtualRC: A Virtual FPGA Platform for Applications and Tools Portability. In *Proceedings of ACM/SIGDA International Symposium on Field Programmable Gate Arrays*, pages 205–208, 2012.

[84] J. Knowles. ParEGO: a hybrid algorithm with on-line landscape approximation for expensive multiobjective optimization problems. *IEEE Transactions on Evolutionary Computation*, 10(1):50–66, Feb 2006.

[85] H. W. Kuhn. Nonlinear programming: a historical view. In *Traces and Emergence of Nonlinear Programming*, pages 393–414. Springer, 2014.

[86] S. Kullback and R. A. Leibler. *Ann. Math. Statist.*, 22(1):79–86, 03 1951.

[87] I. Kuon and J. Rose. Measuring the Gap Between FPGAs and ASICs,. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems.*

[88] M. Kurek, T. Becker, T. C. Chau, and W. Luk. Automating optimization of reconfigurable designs. In *Proceedings of International Symposium on Field-Programmable Custom Computing Machines*, pages 210–213, May 2014.

[89] M. Kurek, T. Becker, and W. Luk. Parametric optimization of reconfigurable designs using machine learning. In *Reconfigurable Computing: Architectures, Tools and Applications*, volume 7806, pages 134–145. Springer-Verlag, Berlin, Heidelberg, 2013.

[90] M. Kurek, T. Liu, and W. Luk. Multi-objective self-optimization of reconfigurable designs with machine learning. In *Proceedings Workshop on Self-Awareness in Reconfigurable Computing Systems*, Sep 2013.

[91] M. Kurek and W. Luk. Parametric reconfigurable designs with machine learning optimizer. In *Proceedings of International Conference on Field-Programmable Technology*, pages 109–112, Dec 2012.

[92] D. Lavenier. Seed-based genomic sequence comparison using a fpga/flash accelerator. *Proceedings of International Conference on Field-Programmable Technology*, 6:41–48, Dec 2006.

[93] C. Leiserson and J. Saxe. Retiming synchronous circuitry. *Algorithmica*, 6(1-6):5–35, 1991.

[94] Q. Liu, G. Constantinides, K. Masselos, and P. Cheung. Combining data reuse with data-level parallelization for FPGA-targeted hardware compilation: A geometric programming framework. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 28(3):305–315, March 2009.

[95] Q. Liu, T. Todman, K. H. Tsoi, and W. Luk. Convex models for accelerating applications on FPGA-based clusters. In *Proceedings of International Conference on Field-Programmable Technology*, pages 495–498, Dec 2010.

[96] D. Lizotte, T. Wang, M. Bowling, and D. Schuurmans. Automatic gait optimization with Gaussian process regression. In *Proceedings of International Joint Conferences on Artificial Intelligence*, pages 944–949, Mar 2007.

[97] D. J. Lizotte. *Practical Bayesian Optimization*. PhD thesis, University of Alberta, Edmonton, Alta., Canada, 2008.

[98] J. Lockwood, A. Gupte, N. Mehta, M. Blott, T. English, and K. Vissers. A low-latency library in FPGA hardware for high-frequency trading (hft). In *High-Performance Interconnects*, pages 9–16, Aug 2012.

[99] Z.-Q. Luo and W. Yu. An introduction to convex optimization for communications and signal processing. *Selected Areas in Communications*, 24(8):1426–1438, Aug 2006.

[100] A. Mametjanov, P. Balaprakash, C. Choudary, P. D. Hovland, S. M. Wild, and G. Sabin. Autotuning FPGA design parameters for performance and power. In *Proceedings of International Symposium on Field-Programmable Custom Computing Machines*, May 2015.

[101] G. Marsaglia and T. A. Bray. A convenient method for generating normal variables. *SIAM Review*, 6(3):260–264, Jul 1964.

[102] C. M. Mcdonald and C. A. Floudas. Global optimization for the phase and chemical equilibrium problem: Application to the nrtl equation. *Computers & Chemical Engineering*, 19:1111–1139, 1994.

[103] M. McKay, R. Beckman, and W. Conover. A comparison of three methods for selecting values of input variables in the analysis of output from a computer code. *Technometrics*, 21(2):55–61, Feb 1979.

[104] J. Mockus. *Bayesian approach to global optimization: theory and applications*. Mathematics and its applications. Kluwer Academic, 1989.

[105] Y. Nesterov. *Introductory lectures on convex optimization : a basic course*. Applied optimization. Kluwer Academic Publ., Boston, Dordrecht, London, 2004.

[106] D. Nguyen-Tuong, M. Seeger, and J. Peters. Model learning with local Gaussian process regression. *Advanced Robotics*, 23(15):2015–2034, 2009.

[107] W. J. Nicholson. Accelerating efficient global optimisation for reconfigurable computing. Master's thesis, Imperial College London, 2015.

[108] J. Nickolls, I. Buck, M. Garland, and K. Skadron. Scalable parallel programming with cuda. *ACM Queue*, 6(2):40–53, Mar 2008.

[109] X. Niu, Q. Jin, W. Luk, Q. Liu, and O. Pell. Exploiting run-time reconfiguration in stencil computation. In *Proceedings of International Symposium on Field-Programmable Custom Computing Machines*, pages 173–180, Aug 2012.

[110] J. Olenšek, T. Tuma, J. Puhan, and Árpád Bűrmen. A new asynchronous parallel global optimization method based on simulated annealing and differential evolution. *Applied Soft Computing*, 11(1):1481–1489, 2011.

[111] C. J. Paciorek and M. J. Schervish. Nonstationary covariance functions for Gaussian process regression. In *Proceedings of Conference on Neural Information Processing Systems*, 2004.

[112] M. Pal. Multiclass approaches for support vector machine based land cover classification. *CoRR*, abs/0802.2411, 2008.

[113] S. J. Pan and Q. Yang. A survey on transfer learning. *IEEE Transactions on Knowledge and Data Engineering*, 22(10):1345–1359, Oct 2010.

[114] K. Pearson. Note on regression and inheritance in the case of two parents. *Proceedings of the Royal Society of London*, 58:240–242, 1895.

[115] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. In *Journal of Machine Learning Research*, volume 12, pages 2825–2830, Jan 2011.

[116] M. Pelikan and M. W. Hauschild. Learn from the past: Improving model-directed optimization by transfer learning based on distance-based bias. *Missouri Estimation of Distribution Algorithms Laboratory, University of Missouri in St. Louis, MO, United States, Tech. Rep*, 2012007, 2012.

[117] C. Pilato, D. Loiacono, A. Tumeo, F. Ferrandi, P. Lanzi, and D. Sciuto. Speeding-up expensive evaluations in high-level synthesis using solution modeling and fitness inheritance. In *Computational Intelligence in Expensive Optimization Problems*, volume 2, pages 701–723. 2010.

[118] J. C. Platt. Probabilistic outputs for support vector machines and comparisons to regularized likelihood methods. In *Advances in Large Margin Classifiers*, pages 61–74. MIT Press, 1999.

[119] R. Pottathuparambil, J. Coyne, J. Allred, W. Lynch, and V. Natoli. Low-latency FPGA based financial data feed handler. In *Proceedings of International Symposium on Field-Programmable Custom Computing Machines*, pages 93–96, May 2011.

[120] J. Pugh and A. Martinoli. Discrete Multi-Valued Particle Swarm Optimization. In *Proceedings of IEEE Swarm Intelligence Symposium*, May 2006.

[121] M. Rabozzi, J. Lillis, and M. D. Santambrogio. Floorplanning for partially-reconfigurable FPGA systems via mixed-integer linear programming. In *Proceedings of International Symposium on Field-Programmable Custom Computing Machines*, pages 186–193, May 2014.

[122] K. Rameshkumar, R. K. Suresh, and K. M. Mohanasundaram. Discrete particle swarm optimization (DPSO) algorithm for permutation flowshop scheduling to minimize makespan. In *Proceedings of First International Conference on Advances in Natural Computation*, pages 572–581, 2005.

[123] M. R. Rapaić, Ž. Kanović, and Z. D. Jeličić. Discrete particle swarm optimization algorithm for solving optimal sensor deployment problem. *Journal of Automatic Control*, 18(1):9–14, 2008.

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

[125] R. Salvador, F. Moreno, T. Riesgo, and L. Sekanina. Evolutionary design and optimization of wavelet transforms for image compression in embedded systems. In *Proceedings of NASA/ESA Conference on Adaptive Hardware and Systems* , pages 177–184, 2010.

[126] R. Salvador, F. Moreno, T. Riesgo, and L. Sekanina. High level validation of an optimization algorithm for the implementation of adaptive wavelet transforms in FPGAs. In *Proceedings of Euromicro Conference on Digital System Design*, pages 96–103, 2010.

[127] M. Schonlau. *Computer Experiments and Global Optimization*. PhD thesis, University of Waterloo, 1997.

[128] T. Schreiber. Measuring information transfer. *Phys. Rev. Lett.*, 85:461–464, Jul 2000.

[129] J. F. Schutte, J. A. Reinbolt, B. J. Fregly, R. T. Haftka, and A. D. George. Parallel global optimization with the particle swarm algorithm. *JOURNAL OF NUMERICAL METHODS IN ENGINEERING*, 61:2296–2315, 2003.

[130] M. Seeger. Gaussian processes for machine learning. *International Journal of Neural Systems*, 14:69–106, 2004.

[131] H. Shah-Hosseini. Problem solving by intelligent water drops. In *Proceedings of IEEE Congress on Evolutionary Computation*, pages 3226–3231, Sep 2007.

[132] H. Shah-Hosseini. Intelligent water drops algorithm: a new optimization method for solving the multiple knapsack problem. In *Journal of Intelligent Computing and Cybernetics*, volume 1, pages 193–212, Jul 2008.

[133] H. Shah-Hosseini. Otsu's criterion-based multilevel thresholding by a nature-inspired metaheuristic called Galaxy-based Search Algorithm. In *Proceedings of World Congress on Nature and Biologically Inspired Computing*, pages 383–388, Oct 2011.

[134] H. Shah-Hosseini. Principal components analysis by the galaxy-based search algorithm; a novel metaheuristic for continuous optimisation. *International Journal of Computer Science Engineering*, 6:132–140, July 2011.

[135] D. Sheffield, M. Anderson, and K. Keutzer. Automatic generation of application-specific accelerators for fpgas from python loop nests. Technical Report UCB/EECS-2012-203, EECS Department, University of California, Berkeley, Oct 2012.

[136] A. Smith, G. Constantinides, and P. Cheung. Area estimation and optimisation of FPGA routing fabrics. In *Proceedings of International Symposium on Field-Programmable Custom Computing Machines*, pages 256–261, Aug 2009.

[137] A. Smith, G. Constantinides, and P. Cheung. Fpga architecture optimization using geometric programming. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 29(8):1163–1176, Aug 2010.

[138] D. J. Smith. *HDL Chip Design: A Practical Guide for Designing, Synthesizing and Simulating ASICs and FPGAs Using VHDL or Verilog*. Doone Publications, 1998.

[139] J. Snoek. *Bayesian Optimization and Semiparametric Models with Applications to Assistive Technology*. PhD thesis, University of Toronto, Toronto, Canada, 2013.

[140] J. Snoek, H. Larochelle, and R. P. Adams. Practical bayesian optimization of machine learning algorithms. In *Proceedings of Conference on Neural Information Processing Systems*, pages 2951–2959. 2012.

[141] J. A. Snyman. *Practical Mathematical Optimization: An Introduction to Basic Optimization Theory and Classical and New Gradient-Based Algorithms*. Springer-Verlag New York, Inc., 2005.

[142] E. Sotiriades, C. Kozanitis, and A. Dollas. FPGA based architecture for DNA sequence comparison and database search. In *Proceedings of IEEE International Parallel and Distributed Processing Symposium*, pages 8 pp.–, Apr 2006.

[143] O. Souissi and A. Abdelhakim. Two exact methods for mapping on heterogeneous CPU/FPGA architectures. In *Proceedings of IEEE International Conference on Networking, Sensing and Control*, pages 520–525, Apr 2013.

[144] N. Srinivas, A. Krause, M. Seeger, and S. M. Kakade. Gaussian process optimization in the bandit setting: No regret and experimental design. In *Proceedings of International Conference on Machine Learning*, pages 1015–1022. Omnipress, 2010.

[145] M. Staniek and K. Lehnertz. Symbolic transfer entropy. *Phys. Rev. Lett.*, 100:158101, Apr 2008.

[146] G. Su. Gaussian process assisted differential evolution algorithm for computationally expensive optimization problems. In *Computational Intelligence and Industrial Application*, volume 1, pages 272–276, Dec 2008.

[147] X. Y. Sun, D. Gong, and S. Li. Classification and regression-based surrogate model-assisted interactive genetic algorithm with individual's fuzzy fitness. In *Proceedings of conference on Genetic and evolutionary computation*, pages 907–914, Mar 2009.

[148] K. Swersky, J. Snoek, and R. P. Adams. Multi-task bayesian optimization. In *Advances in Neural Information Processing Systems*, volume 26, pages 2004–2012. Curran Associates Inc., 2013.

[149] E.-G. Talbi. *Metaheuristics: From Design to Implementation*. Wiley Publishing, Hoboken (N.J.), 2009.

[150] Y. Tang, J. Chen, and J. Wei. A surrogate-based particle swarm optimization algorithm for solving optimization problems with expensive black box functions. *Engineering Optimization*, 45(5):557–576, 2013.

[151] D. Thomas and W. Luk. Non-uniform random number generation through piecewise linear approximations. In *Proceedings of International Symposium on Field-Programmable Custom Computing Machines*, pages 1–6, Aug 2006.

[152] D. B. Thomas, W. Luk, P. H. Leong, and J. D. Villasenor. Gaussian random number generators. *ACM Computer Survey*, 39(4), Nov 2007.

[153] X. Tian and K. Benkrid. Design and implementation of a high performance financial monte-carlo simulation engine on an FPGA supercomputer. In *Proceedings of International Conference on Field-Programmable Technology*, pages 81–88, Dec 2008.

[154] M. E. Tipping and A. Smola. Sparse Bayesian learning and the relevance vector machine. In *Journal of Machine Learning Research*, volume 1, pages 211–244, Sep 2001.

[155] A. H. Tse, G. C. Chow, Q. Jin, D. Thomas, and W. Luk. Optimising performance of quadrature methods with reduced precision. In *Reconfigurable Computing: Architectures, Tools and Applications*, volume 7199, pages 251–263. 2012.

[156] C. C. Tutum, K. Deb, and I. Baran. Constrained efficient global optimization for pultrusion process. *Materials and Manufacturing Processes*, 30(4):538–551, 2015.

[157] F. Van Den Bergh. *An analysis of particle swarm optimizers*. PhD thesis, University of Pretoria, South Africa, 2002.

[158] J. Villemonteix, E. Vazquez, and E. Walter. An informational approach to the global optimization of expensive-to-evaluate functions. *Journal of Global Optimization*, 44(4):509–534, 2009.

[159] J. Whitaker. *The Electronics Handbook, Second Edition*. Electrical Engineering Handbook. CRC Press, 2005.

[160] B. Xia, Z. Ren, and C. Koh. Comparative study on kriging surrogate models for metaheuristic optimization of multidimensional electromagnetic problems. *IEEE Transactions on Magnetics*, 51(3):1–4, March 2015.

[161] W. Xu, K. Xu, and X. Xu. A novel placement algorithm for symmetrical FPGA. In *Proceedings of IEEE International Conference on ASIC*, pages 1281–1284, Oct 2007.

[162] C.-L. Yeh and R. Chin. Constrained optimization for image restoration using nonlinear programming. In *Proceedings of IEEE International Conference on Acoustics, Speech and Signal Processing*, volume 10, pages 676–679, Apr 1985.

[163] P. B. N. Yew S. Ong and A. J. Keane. Evolutionary optimization of computationally expensive problems via surrogate modeling. *AIAA Journal*, 41(4):689–696, 2003.

[164] L. A. Zager and G. C. Verghese. Graph similarity scoring and matching. *Applied Mathematics Letters*, 21(1):86–94, 2008.

[165] W. Zhang, V. Betz, and J. Rose. Portable and scalable FPGA-based acceleration of a direct linear system solver. *ACM Transactions on Reconfigurable Technology and Systems*, 5(1):1–26, Mar 2012.

[166] Z. Zhang, Y. Fan, W. Jiang, G. Han, C. Yang, and J. Cong. Autopilot: A platform-based esl synthesis system. In P. Coussy and A. Morawiec, editors, *High-Level Synthesis*, pages 99–112. Springer Netherlands, 2008.

[167] D. Zwillinger and S. Kokoska. *CRC standard probability and statistics tables and formulae*. CRC, 1999.