UNIVERSITY OF CALIFORNIA SAN DIEGO

Automatic Optimization of System Design for 3D Mapping of Archaeological Sites

A dissertation submitted in partial satisfaction of the
requirements for the degree of Doctor of Philosophy

in

Computer Science

by

Quentin K. Gautier

Committee in charge:

Professor Ryan Kastner, Chair
Professor Manmohan Chandraker
Professor Kamalika Chaudhuri
Professor Truong Nguyen
Professor Laurel Riek

2019

The Dissertation of Quentin K. Gautier is approved and is acceptable in quality and form for publication on microfilm and electronically:

Chair

University of California San Diego

2019

DEDICATION

This work is dedicated to my parents who have always pushed me on a path to education, and supported me during my Ph.D. despite being so far from home.

To all of my family and my friends.

EPIGRAPH

When you have eliminated the impossible, whatever remains,
however improbable, must be the truth.

Sir Arthur Conan Doyle

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGEMENTS

I would like to acknowledge Professor Ryan Kastner as my advisor and the chair of my committee. He provided me with the support, funding, guidance, and opportunities that made this work possible. I would also like to thank my committee: Professor Manmohan Chandraker, Professor Kamalika Chaudhuri, Professor Truong Nguyen, Professor Laurel Riek.

All of my co-authors, collaborators, colleagues, and friends have been a great help in all of my projects. A special thanks goes to Alric Althoff who has helped me so many times and taught me so much, as well as Pingfan Meng who has provided me with the foundations of much of my work, Janarbek Matai who gave me my first project, and Alireza Khodamoradi for his help as a T.A. and a friend. Finally, I would like to thank all members of Kastner Research Group and Engineers for Exploration, who all have been amazing.

Chapter 1, in part, has been submitted for publication of the material as it may appear in the Journal of Cultural Heritage Management and Sustainable Development. Gautier, Quentin; Garrison, Thomas; Rushton, Ferrill; Bouck, Nicholas; Lo, Eric; Tueller, Peter; Schurgers, Curt; and Kastner, Ryan. The dissertation author was the primary investigator and author of this material.

Section 2.3, in part, is a reprint of the material as it appears in the International Conference on Field-Programmable Technology (FPT) 2014. Gautier, Quentin; Shearer, Alexandria; Matai, Janarbek; Richmond, Dustin; Meng, Pingfan; and Kastner, Ryan. The dissertation author was the primary investigator and author of this paper.

Chapter 2, in part, is a reprint of the material as it appears in the International Conference on Application-specific Systems, Architectures and Processors (ASAP) 2019. Gautier, Quentin; Althoff, Alric; and Kastner, Ryan. The dissertation author was the primary investigator and author of this paper.

Chapter 3, in part, is a reprint of the material as it appears in the International Conference on Field-Programmable Technology (FPT) 2016. Gautier, Quentin; Althoff, Alric; Meng, Pingfan; and Kastner, Ryan. The dissertation author was the primary investigator and author of

this paper.

Chapter 4, is coauthored with Althoff, Alric; Meng, Pingfan; and Kastner, Ryan. The dissertation author was the primary author of this chapter.

Chapter 5, in part, has been submitted for publication of the material as it may appear in the 28th ACM/SIGDA International Symposium on Field-Programmable Gate Arrays (FPGA), 2020. Gautier, Quentin; Althoff, Alric; and Kastner, Ryan. The dissertation author was the primary investigator and author of this paper.

VITA

2010	M.S. in Computer Science, Institut National des Sciences Appliquees de Rennes, France
2013-2019	Research Assistant, Department of Computer Science And Engineering University of California San Diego
2014-2019	Teaching Assistant, Department of Computer Science And Engineering University of California San Diego
2019	Ph.D. in Computer Science, University of California San Diego

PUBLICATIONS

Quentin Gautier, Alric Althoff and Ryan Kastner. "FPGA Architectures for Real-Time Dense SLAM." In 2019 IEEE 30th International Conference on Application-specific Systems, Architectures and Processors (ASAP), IEEE, 2019.

Quentin Gautier, Alric Althoff, Pingfan Meng, and Ryan Kastner. "Spector: An opencl fpga benchmark suite." In 2016 International Conference on Field-Programmable Technology (FPT), pp. 141-148. IEEE, 2016.

Pingfan Meng, Alric Althoff, Quentin Gautier, and Ryan Kastner. "Adaptive threshold non-pareto elimination: Re-thinking machine learning for system level design space exploration on FPGAs." In Proceedings of the 2016 Conference on Design, Automation & Test in Europe, pp. 918-923. EDA Consortium, 2016.

Quentin Gautier, Alexandria Shearer, Janarbek Matai, Dustin Richmond, Pingfan Meng, and Ryan Kastner. "Real-time 3d reconstruction for fpgas: A case study for evaluating the performance, area, and programmability trade-offs of the altera opencl sdk." In 2014 International Conference on Field-Programmable Technology (FPT), pp. 326-329. IEEE, 2014.

ABSTRACT OF THE DISSERTATION

Automatic Optimization of System Design for 3D Mapping of Archaeological Sites

by

Quentin K. Gautier

Doctor of Philosophy in Computer Science

University of California San Diego, 2019

Professor Ryan Kastner, Chair

Deeply buried within the jungle of Central America lie the remains of the ancient Maya civilization. In order to reach these old ruins, archaeologists dig tunneling excavations spanning tens of meters underground. Unfortunately, most of these excavations must be closed for conservation purposes, and it is therefore crucial to document these findings as precisely as possible. Many solutions exist to create a 3D scan of these environments, but most of them are too costly, difficult to deploy, or do not provide real-time feedback.

A possible solution to create a 3D mapping system overcoming these problems is to use Simultaneous Localization And Mapping (SLAM) algorithms, combined with low-cost sensors and low-power hardware. However, the combined complexity of software design and hardware

design represents an immense challenge to implement a system optimized for all requirements. The vast pool of possible designs and the multiple, often conflicting objectives contribute to produce design spaces too complex to be explored manually.

In this thesis, we explore the complexity of designing SLAM applications using various types of hardware. First, we manually evaluate SLAM algorithms for 3D mapping, and specifically optimize one SLAM algorithm on an FPGA hardware. Then we expand our exploration to a larger space of designs, and generalize this problem to the design of all complex applications that require a lengthy evaluation time.

We develop several learning-based methods to help designers finding the best combinations of optimizations that maximize multiple objectives. By using a smart sampling algorithm, and removing the need of selecting a specific regression model, we can help users largely decrease the number of designs to evaluate before reaching an optimal architecture.

Introduction

A Scanning System for Archaeology

When searching for the remains of the ancient Maya civilization in Mesoamerica, archaeologists dig narrow tunnels to excavate old structures that have been outgrown by the jungle over the centuries. These excavations can reveal important information about the Maya culture, their traditions and their rituals. Unfortunately, most excavations are temporary, as they expose fragile structures to the outside, and need to be closed eventually. As a result, documenting these sites as precisely as possible – and sometimes as fast as possible – is a crucial task. In order to document these tunnels, we can design a system to collect three-dimensional data to save these environments as 3D models. However, the design of such a system is not trivial. It needs to be portable, simple to use, low-power, accurate, fast, and it must fit within the budget of the excavation site. An attractive solution to this problem is to leverage Simultaneous Localization And Mapping (SLAM) algorithms [35]. This class of algorithms can be combined with inexpensive sensors to build a digital model of a scene in real-time. But the large number of existing algorithms and sensors leaves designers with a myriad of options (e.g., [61, 78, 65, 66, 67, 72, 58, 59, 39, 54, 82]...) for which the influence on each optimization goal can be difficult to anticipate. Sometimes, the relationship between design options and the objective result is fairly straightforward. For example, when utilizing a lower-cost sensor, one can expect a lower overall accuracy compared to more expensive hardware. That relationship can become more complicated very quickly. Typically, a designer could choose a stereo camera as a scanning equipment to improve ease of use and scanning speed when switching from outdoor environments to indoor. However, the loss

of accuracy might actually slow down the user to avoid loss of tracking. This problem can be solved by using an algorithm supporting fast recovery, but that algorithm might provide you with a sparser 3D map, and increase the overall complexity of the system.

In this situation, a designer must carefully think about the impact of each design choice, and eventually implement a small subset of all possible solutions. Testing these designs, evaluating the results, and refining the system is a very slow process, especially when creating a full end-to-end solution targeting a very specific environment that cannot easily be reproduced in a lab. We are now left with a question: how do we design such a complex system without having to test all possible solutions, but still quickly converging toward an optimal system with respect to multiple conflicting objectives?

System Design Optimization

The design of complex hardware and software systems is generally a long, iterative process, in which one or multiple domain experts build and optimize different subsystems based on their knowledge, and design methodologies. The field of software engineering has refined design methodologies over the last few decades [69] with good results, but at the same time, the software complexity has increased dramatically [70]. In 1983, Lampson [68] distinguishes the design of computer systems from the design of algorithms, citing more complex requirements, more internal structure, and a less clear measure of success for physical systems. However, algorithms have become more complex over the years, and as a result, can be considered in some cases as difficult to build as a hardware system. The field of computer vision is a good example of how complex algorithms can become. Computer vision are often composed of multiple submodules [49], and the methods of evaluation of such systems have not always been clearly defined [49, 125, 124, 105], have continuously evolved, and continue to change regularly. New benchmarks and metrics are still being developed, based on the ever-changing requirements of the field [17].

In parallel to the evolution of software algorithms, hardware design has also become more complex. With Moore's law approaching to an end [112], the hardware architecture of processors needs to be revised to keep improving performance. At the same time, Graphics Processing Units (GPU) have become increasingly popular, first driven by the video games market, then by scientific computing and machine learning training. More recently, there has been a trend toward custom hardware design [7], either through Application-Specific Integrated Circuits (ASIC) or Field-Programmable Gate Arrays (FPGA), in order to push the performance of algorithms without compromising power consumption. Hardware design has also become more approachable by a larger set of programmers, thanks to the High-Level Synthesis (HLS) tools [8]. With this growing diversity of available hardware and software tools, more designers conceive systems simultaneously on hardware and software [32].

The ever growing complexity of hardware and software design has led to an optimization problem. Each component of a system can potentially be tuned and adjusted, which leads the entire system to produce a different result. Depending on the size and complexity of the design, the behavior of these parameters is not always easy to predict, and multiple parameters can interact in very unpredictable ways. Additionally, the requirements of a system can be conflicting, such that improving one objective tends to deteriorate a second objective (e.g., accuracy vs. performance). A large system can comprise of hundreds of parameters with thousands or millions of possible combinations. The problem of finding the most optimal designs among all possible combinations is called the design space exploration problem. Design space exploration has been studied, and several solutions have been proposed over the years [10, 96, 60, 114, 20, 50, 101, 77]. Certain solutions are based on systems that can be evaluated quickly (i.e., sub-second evaluation), however most hardware design problems can take hours or days to evaluate, and certain software algorithm suffer from similar issues. Other solutions either require specific knowledge of the system, or require fine-tuning of hyperparameters.

Dissertation Outline

In this dissertation, we study the problems and tradeoffs involved in the design of complex SLAM applications, then expand these problems to the design of hardware-accelerated applications. We provide solutions to accelerate and automate this process through design space exploration techniques. Specifically, we show that **learning-based design space exploration tools can automatically find the most optimized software and hardware designs on multiple objectives, with minimal inputs from designers, no training, and only a small amount of design evaluations**.

First, we manually implement several solutions to build a tunnel mapping system for archaeology, and evaluate them one by one. Then, we focus on solutions that can potentially reduce the power footprint of the system, by implementing a 3D scanning algorithm on FPGA. When creating an architecture for FPGA, the implementation can take multiple hours to obtain a final design. Yet the number of optimization parameters is still very large, and a domain expert must carefully analyze all the possible options to pick the most optimal ones. We analyze these design spaces to better understand the impact of various design parameters. Following this application-specific analysis, we generalize the problem to other types of applications by creating benchmarks to explore various FPGA design spaces. We then focus on improving design space exploration tools targeted at FPGA hardware, and finally conceive a general-purpose design space exploration framework for optimizing multi-objective applications.

This thesis is organized as follows:

Chapter 1 presents our system design and evaluation to map tunneling excavations in archaeological sites. We manually choose a set of sensors and algorithms that fit low-cost requirements, and measure the accuracy of different solutions to better understand the tradeoffs of each design choice.

Chapter 2 focuses on reducing the power utilization of mapping algorithms by implementing multiple FPGA architectures. In this chapter, we manually tune the algorithms, but

also provide very parameterized designs that we analyze to better understand the influence of parameters over optimization goals.

Chapter 3 generalizes the FPGA design problem by creating a large set of applications, each heavily parameterized. We construct several design spaces that contains thousands of unique architectures. We provide an analysis of certain spaces to show that intelligent algorithms are necessary to efficiently find the optimal sets of parameters.

Chapter 4 improves existing machine learning frameworks that perform design space exploration. This chapter focuses on FPGA designs, and devises a technique that can leverage data from GPU designs to help prune the search of optimal points within the design spaces.

Chapter 5 presents the design of an active learning framework that focuses on quickly finding Pareto optimal designs for any system design. The framework is capable of converging rapidly toward optimal solutions without making any assumptions on the type of system, and can choose the best prior to build a good regression model. It does not need expert input, or training over similar applications. We demonstrate the use of this framework on FPGA design problems, but also on a computer vision software application.

Chapter 1

Low-Cost 3D Scanning Systems for Cultural Heritage Documentation

1.1 Introduction

Most of the remains of the ancient Maya civilization are currently buried under the dense jungle of Central America. A myriad of temples, platforms, burial chambers, and other structures lie underground. Their excavation can provide invaluable information about the ancient Maya culture and lifestyle. In this region, a common excavation strategy is to dig tunnels leading to major parts of these structures for observation and analysis. Unfortunately, not all of these tunnels can be kept open to avoid the degradation of the exposed structures. Moreover, even the open portions of the excavations are located in a remote area – and sometimes on arduous terrain –, making it difficult to access for most people. This leads to the crucial work of documenting the excavation for historical records, off-site analysis, and public outreach.

Digital documentation of cultural heritage is becoming commonplace at archaeological sites where high-resolution, remotely-sensed data provide a more accurate record than traditional analog recording of excavation contexts, architecture, and ancient monuments (e.g. [41]). The collection of this data can come at great financial cost, as with terrestrial LiDAR (light detection and ranging) systems, or be time consuming during post-processing, as with photogrammetry. In some archaeological contexts these are not effective solutions for documenting cultural heritage. Landscapes with extensive damage from looting and other destructive activities require too much

field time for sensitive, expensive instruments and a lack of real-time results can leave doubt about whether the sensors effectively documented the targets.

Large-scale aerial LiDAR acquisitions in Belize [23] and Guatemala [19] reveal thousands of previously undetected structures created by the ancient Maya civilization (1000 BCE–1519 CE), but also illuminates the extent to which unchecked, illicit looting has damaged this landscape. Ground-truthing of aerial LiDAR data would benefit from the ability to simultaneously acquire 3D documentation of looted contexts to more accurately quantify damage in cultural heritage management assessments. The recent wanton destruction of archaeological sites, like Palmyra, by the Islamic State of Iraq and Syria (ISIS) presents an extreme case of a high-risk zone where recovery efforts could have benefited from a real-time, rapid documentation system to provide damage assessment estimates. Unfortunately, such incidents are not unique, and archaeology would benefit from a rapid 3D scanning sensor to record damage and plan recovery.

The recent growth of low-cost, consumer-grade depth sensors such as Microsoft Kinect or Intel Realsense, has led to an increase of algorithms capable of using these sensors to create a geometrical representation of the surrounding space. These methods have the benefit of being easy to deploy, fast to operate, and produce real-time results that can be visualized directly on the field. The right combination of low-cost sensors and open-source algorithms can provide an accessible scanning solution to untrained users, while being fast to deploy in situations where the site must be closed shortly after being excavated, and inexpensive to replace in case of damages due to difficult field conditions.

However, in order to build such a system, one must carefully choose between all available options in terms of software and hardware, in order to maximize the accuracy of the result, and maximize the usability of the entire system.

In this chapter, we evaluate multiple sensor options by selecting popular depth cameras that can be used out-of-the-box without technical knowledge of the hardware. We have selected several open-source Simultaneous Localization And Mapping (SLAM) algorithms, and config-

ured them to work with our selected sensors without any major modification. We have built a prototype backpack system to test the various hardware/software combinations in the field.

While this new system is not as spatially accurate as more expensive or time-consuming methods, we believe that there are sufficient examples of extensively looted archaeological landscapes and zones with high risk for cultural heritage destruction that a rapid, cost-effective 3D documentation system such as the one presented here would be an asset in cultural heritage management. The system was tested in ancient Maya archaeological contexts and refined in mud caves near the University of California, San Diego, where we developed the prototype.

1.2 Background

Our prototype is designed to efficiently document underground or other restricted spaces such as those presented by damaged ancient architecture. There exist multiple ways of collecting three-dimensional data to reconstruct an underground or indoor environment, that we can group into three main categories: 1) scans using terrestrial LiDAR, 2) scans using photogrammetry, and 3) scans based on Simultaneous Localization And Mapping (SLAM) algorithms with lower-cost sensors.

1.2.1 LiDAR

Terrestrial LiDAR scanners are high precision devices that can capture the geometry of a space by sending lasers in a wide area around the scanner. These devices are often used for cultural heritage documentation due to their very high accuracy. This type of documentation is popular in Maya archaeology [41, 27], which means that there is good, high-resolution baseline 3D data to compare against other sensors and methods. While the 3D scans generally get millimeter accuracy, the process of collecting the data over extensive excavation sections is very long. Furthermore, the final result is obtained after a lengthy post-processing step that cannot be performed directly in the field, preventing a quick visualization of the current scan. Finally, the cost of high-quality laser scanners is often a barrier to large-scale deployment, or deployment

in difficult environments (e.g., narrow, dusty tunnels) where the equipment is at risk of being damaged.

1.2.2 Photogrammetry

"Structure-from-motion" photogrammetry is a general technique that uses computer vision algorithms to reconstruct a 3D shape from a set of 2D pictures. It is a very accessible solution to scan different types of environments; the user only needs a digital camera, and potentially a good light source. This technique also provides good results for cultural heritage documentation [120, 37]. The largest drawback is the computational time required to process the data. Due to this limitation, it can be difficult to evaluate the results in the field. Additionally, the computation time grows very rapidly with the number of photos, and that number grows quickly with the size of the area to scan, which renders the scan of an entire excavation very difficult. The sometimes-cramped conditions of archaeological contexts with fragile cultural heritage monuments can also be a deterrent to photogrammetric methods that require the photographer to be in close proximity to sensitive features for extended periods of time.

1.2.3 Simultaneous Localization And Mapping

Simultaneous Localization And Mapping (SLAM) algorithms are a class of algorithms – first defined in 1986 [104] – focused on localizing an agent within an unknown environment while simultaneously building a map of this environment [35]. The goal of SLAM is to provide an update of the position and/or a map in real-time. The definition of real-time varies with the application, but in our case, we target an update around the same speed as camera sensors, i.e., 30 Hz. We are particularly interested in the use of visual SLAM algorithms, utilizing visual sensors, but SLAM works with any number and combinations of sensors ([61, 78, 72, 39, 54, 82]).

In parallel with the development of SLAM algorithms, many low-cost visual and depth sensors have been commercialized in the past few years. The Microsoft Kinect and the Intel Realsense are two examples of such sensors, which combine an RGB camera with a depth sensor

that provides geometrical information about the scene.

The combination of SLAM algorithms and low-cost sensors yields a good solution to the problem of obtaining quick 3D scans of a scene. However, the quality of results is unknown and can be difficult to assess for multiple reasons. First, there is a wide variety of SLAM algorithms available, all using different methods to achieve the best quality of results. Second, each sensor has its own specificity, and specifically all sensors present a certain degree of noise that needs to be mitigated by the algorithms. Finally, many algorithms are developed in a similar, well-lit environment, but few are tested in the field where the condition are often less than ideal to maintain a good quality of tracking. We tested our prototype in Guatemala where archaeological excavations at a Classic Maya (250–1000 CE) site present a challenging environment due to the lack of good lighting, and narrow tunnels without connecting loops.

Several studies actually compare and evaluate SLAM algorithms. Zennaro and colleagues [122] compared two low-cost depth cameras in terms of noise, and Krafts team [62] evaluated these cameras on trajectory estimation. Huletski and colleagues [53] compared multiple SLAM algorithms on a standard dataset. The common point of these studies is the use of an indoor, office-like environment. Though there has also been extensive evaluation of SLAM in outdoor environments for autonomous driving [16]. Most of the evaluations are driven by the availability of data, typically through standard benchmarks with provided ground truth (e.g., [105, 45]).

Other work studies the usage of various handheld or robot-mounted sensors for 3D scanning of cultural heritage sites. Various studies present the evaluation of a handheld LiDAR scanning system in different field environments ([129, 36, 33, 89]). This system constitutes an improvement over larger terrestrial LiDARs in terms of usability, but remains a costly piece of equipment along with closed-source, commercial software. Erica Nocerino [89] and Masiero and colleagues [76] both evaluated a commercial backpack mapping system, providing a good mobile sensing solution, but also containing costly sensors. We present a study of multiple SLAM algorithms with multiple consumer-grade depth cameras, (see [71] for a similar study),

in a more constrained environment, and comparing more recent and complex SLAM algorithms.

1.3 Methodology Overview

In order to evaluate the digital reconstruction accuracy of SLAM algorithms, we collected data by using multiple combinations of sensor hardware and application software. We performed the data collection at different underground locations, each with its own unique conditions. In addition to the SLAM data collection, we also gathered data from multiple LiDAR scans at the same locations to provide a reference for the various 3D models. All the LiDAR scans were taken with ample overlap to allow for a very precise manual alignment, thus providing us with ground-truthed 3D models which can be compared against the results of the real-time SLAM-based 3D scanning.

1.3.1 Data Collection Sites

We evaluated our prototype inside active excavation tunnels at the Maya archaeological site of El Zotz, Guatemala. Most of our experiments were performed in the Pyramid of the Wooden Lintel (Structure M7-1), an approximately 20-meter-tall structure that was extensively looted in the 1970s before archaeologists began extending the existing tunnels in 2012 to salvage information from the structure [40, 52]. This excavation is large enough to create a good test environment, and has been thoroughly scanned by LiDAR. Furthermore, the looted section of the tunnel can be considered representative of the types of contexts where our prototype scanner would be most useful.

We performed a second set of experiments in the Arroyo Tapiado Mud Caves system [51] located within the Anza-Borrego State Park near San Diego, California. These mud caves provided a controlled environment in a local context where we could run multiple similar scans without the constraints of an active excavation or the expense of working through developmental trial and error in an international field context where some resources are limited. While these natural caves are generally larger than a typical archaeological tunnel, they still present a similar

setup in terms of scanning challenges. We particularly chose to operate our system in locations with sharp turns to emulate the shape of excavations.

1.4 Data Acquisition

Our goal was to measure the performance of SLAM-based scanning techniques in a realistic scenario. We made hardware and software decisions based on a set of requirements aimed at providing archaeologists with a usable system in the field. These requirements consist of designing a system that is lightweight, simple, usable by a single non-technical user, and is easily manipulated inside a tight and dark tunnel environment. In addition, the system must provide its own power, be self-contained (no external connection), and can be transported to and operated in a difficult environment (hot, humid, dusty, etc.). We also wanted to build a device based on readily available and low-cost components and materials, and design software tools that are open-source.

Our prototype consists of a backpack containing a laptop, and an external tablet with a light panel and 3D sensors. The backpack system is presented in Figure 1.1 and described in detail below. The laptop is the most expensive component, in order to run the real-time software needed for the scanning.

1.4.1 Hardware Setup

The first part of our scanning device consists of a backpack modified with a solid frame that can handle a large laptop while keeping a good airflow to prevent overheating. In the same backpack, we keep several lithium polymer (LiPo) batteries to let the computer run for several hours, and provide power to the sensors and lights. The second part of the device is a custom frame supporting a tablet facing the user, a light panel facing forward, and a mount for a 3D sensor facing forward. The frame possesses handles for better manipulation in difficult areas, and to give the user flexibility to scan at different angles. The tablet provides feedback from the laptop, and can display the status of the scanning operation. The light panel is designed

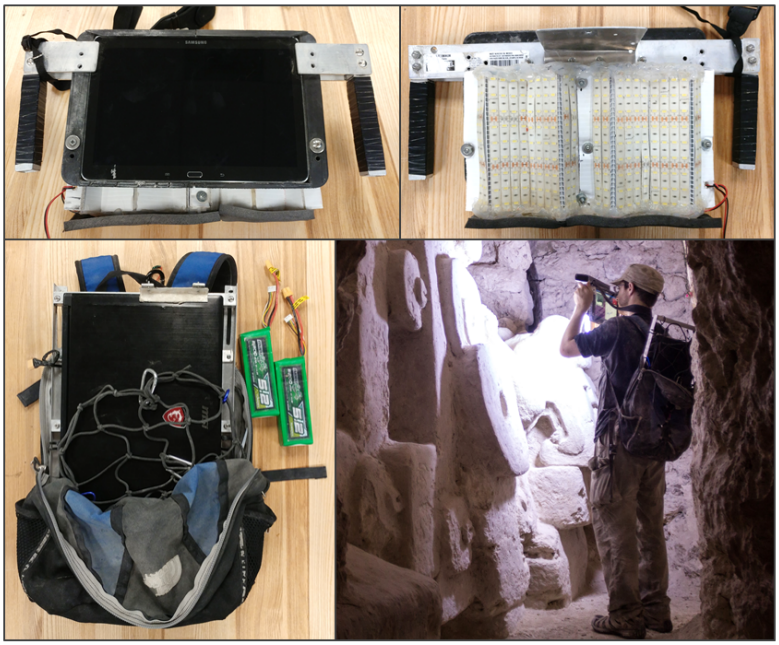


Figure 1.1. Scanning backpack prototype. On top is the handheld device, containing a light panel, a sensor mount, and a tablet for visualization. On the bottom left is the backpack containing a laptop and a metal frame for better cooling. The bottom right picture illustrate the use of the backpack inside the excavation.

to provide enough brightness in a completely dark tunnel area, while limiting the creation of hard shadows that can cause issues with the SLAM algorithms. It consists of several strip of small LEDs that emit a bright white and diffuse light, sufficient to cover the field of view of our cameras. The sensor mount is designed to easily swap the sensor in use. Both parts of our hardware design are described in more details in our repository [1].

SLAM algorithms are optimized for a certain set of sensors. A very popular solution is to use RBG cameras as they are generally low-cost, light, and small. However, while simple RGB cameras are good for tracking the position of a device, e.g., a drone or robot, for 3D reconstruction purpose, we chose to use RBG-D cameras. RGB-D cameras are active or passive cameras that also generate 3D depth information alongside the regular RGB feed. To this purpose, they employ different technologies: structured light, time-of- flight, or stereo vision. In our case, we tested RGB-D cameras covering these three types of technologies, all based on a near-infrared

[1] https://github.com/UCSD-E4E/maya-archaeology

Table 1.1. RGB-D cameras used in our experiments

	Technology	Notes
Microsoft Kinect v1	Structured light	
Microsoft Kinect v2	Time-of-flight	Global shutter
Intel Realsense ZR300	Active stereo	Works outdoors
Intel Realsense D435	Active stereo	Works outdoors

spectrum of light. The cameras that we use are summarized in Table 1.1. Two of these cameras can potentially operate in daylight, while the others only work in an indoor or underground environment only.

1.4.2 Software Setup

For the 3D reconstruction in real-time, we need software that can process the data from the RGB-D camera to create a three-dimensional map of the environment, while providing visual feedback to the user. The visual feedback is important to track the status of the algorithms running, e.g., whether the algorithm has lost track of its previous trajectory, in which case the user needs to come back to a previous position to restart the scanning. It is also useful to determine how much area has been scanned, and make a decision on the next area to process.

The research literature is abundant on the topic of camera-based SLAM algorithms, but not all provide an end-to-end application that meets our requirements. We chose a few popular open-source systems that can easily be setup and used with the hardware described earlier. We chose the following algorithms:

- RTAB-Map [65, 66, 67]. RTAB-Map is a SLAM framework that provides a dense 3D reconstruction of the environment from different possible types of sensors. The framework presents many features and parameters to adjust the speed and accuracy of the scan. The total number of parameters is very high, and testing all the possible options is extremely time-consuming and not feasible on the field with limited time and resources. The software comes with a good default set of parameters that we use in our experiments.

- ORB-SLAM [82]. ORB-SLAM is another popular open-source SLAM algorithm. The open-source version of ORB-SLAM 2 provides a real-time tracking based on a camera input with optional depth sensing. The application has very little parameters and focuses on delivering an out-of-the-box functional tracking solution. The drawback of this algorithm is that it only saves a sparse map of the environment. In order to create a full 3D reconstruction, we need to apply some post-processing to the output data.

- InfiniTAM v2 and InfiniTAM v3 [58, 59]. InfiniTAM is an application that focuses on RGB-D-based 3D reconstruction. It creates a dense map of the environment in real-time by leveraging the GPU compute capabilities of the computer to accelerate the scanning. The difference between the two versions is mainly a loop closure feature introduced in v3 that provides a global consistency to the map, but sometimes at the cost of stability in our experiments.

All SLAM algorithms are subject to drift. Drift is introduced by the accumulation of small errors during the reconstruction process, and is usually the main reason of low accuracy in the resulting 3D models. For this reason, tracking is an important step for SLAM algorithms, along with loop closure. Loop closure enables a global optimization of the map when the user creates a loop in the scanning trajectory and re-visits a place previously scanned. RTAB-Map and ORB-SLAM both implement tracking based on visual features, i.e., points of interest within the RBG images, and have loop closure sub-modules. InfiniTAM implements a tracking algorithm based on 3D depth data, and only introduces loop closure in v3.

1.4.3 Acquisition Methodology

In all of our experiments, we used the backpack system described in Section 1.4.1, and collected data with all the cameras available at the time of the experiment. For each of the experiments, we ran one of the SLAM algorithms described in Section 1.4.2, but we also recorded all the sensor data on the computer drive. This setup allowed us to replay the collected

data through other algorithms in a very similar condition as the field experiment.

The first data collection at the El Zotz archaeological site happened in 2014, when we extensively scanned multiple excavations using a FARO LiDAR scanner [41]. We use these LiDAR scans as ground truth reference models. In 2016, we started to run our SLAM experiments with various combinations of sensors and algorithms. The main results that we analyze in Section 1.6 were generated from large-scale scans covering about 45 m of the excavation tunnel in the Pyramid of the Wooden Lintel (Str. M7-1). We used the Kinect v1, Kinect v2, and ZR300 cameras, and ran the ORB-SLAM algorithm. Most of these results start from the entrance of the tunnel, proceed to the end of the active excavation, turn around to reach the entrance again, and turn around one last time to force the algorithm to perform a loop closure. This path goes through a series of steep architectural tiers from one of the earlier pyramidal substructures, including small ladders to transverse them. This renders the scans difficult in that area, but it is representative of the difficult field conditions that may be encountered during cultural heritage documentation.

Outside of the field seasons, from 2016 to 2018, we chose the Arroyo Tapiado Mud Caves site as a control site to refine our prototype. We selected a section of the *Carrey* cave, presenting several sharp turns. We scanned this whole section with multiple hardware/software setups, by using a very similar scanning walking pattern each time. We experimented with both Kinect cameras, the ZR300 and D435 cameras, and used ORB-SLAM and RTAB-Map. We also provided an experimental setup with two Kinect cameras simultaneously scanning the environment with the RTAB-Map framework. On each experiment, we also recorded the sensor data to possibly be replayed later through other algorithms.

1.5 Data Processing

Our goal was to measure the quality of results of various combinations of sensors and algorithms. In this context, the quality of the result corresponds to the difference between each

sensor and algorithm combination as compared against a reference ground-truth model, which is considered highly accurate. In this case, we consider the LiDAR models to be extremely accurate, as they consist of dozens of 360 degrees scans carefully registered together [41]. However, quantifying the difference between our experiment models and the ground truth reference can be challenging due to multiple factors affecting the output: sensor noise/accuracy, algorithm noise/accuracy, resolution of the output, etc. We chose to measure model quality with two different methods: one comparing the models directly, and one comparing the trajectory.

1.5.1 Measuring point cloud difference

The most obvious method to evaluate 3D scanning output against an available reference model is to overlay them together and observe the difference. However, a manual over- lay can be difficult and introduce further error that is not related to the scanning method. We solve this problem by aligning the point clouds through rigid registration. We use the Iterative Closest Point (ICP) [9] method from the CloudCompare [47] software to find the best alignment between each SLAM result and the ground truth LiDAR scan. From these aligned models, we can compute a point-to-point difference [25] and use it to colorize the points. This colored model highlights the area with high difference compared to the reference. While the colored model can provide valuable information, it requires a visual inspection. We also summarize all the point-to-point differences by calculating the Root Mean Squared Error (RMSE) which provides the average Euclidean distance error between points. The RMSE can be used as a metric for a quick comparison between results.

This method suffers from one major issue: all the types of error are averaged. First, the registration provides the best alignment that minimized the average error over all points. This means that the location of areas with high error is not always accurate (larger error tends to appear on edges of the model). The statistics (min, max, RMSE, etc.) are still meaningful, however they include multiple types of error, including sensor errors and software errors. This is why a visual inspection and a second quality metric are needed.

Measuring trajectory difference

When evaluating and comparing SLAM algorithms, most of the literature focuses on comparing the trajectories [105, 34, 45, 18]. A trajectory is defined as the position and rotation of the camera center at each scan (or at each *key* scan, depending on the algorithm) in the 3D space. The trajectory is generally considered as the most important output of the SLAM algorithm. If this output is accurate, then sensor 3D data can be joined together into a coherent model. Sensor noise can be reduced or averaged by using techniques that rely on an accurate estimate of the camera positioning. Several metrics exist to evaluate the quality of the trajectory with respect to a ground truth.

In our experiments however, we do not possess a ground-truth trajectory. Instead, we recover it by incrementally registering each camera scan to the ground truth LiDAR model. The process starts by a manual alignment of the first scan, which is followed by a fine-tuned alignment from an ICP algorithm. Each following scan is then aligned using ICP as well. We reduce the amount of sensor error by removing data past 3 m of the camera (as the error generally increases with the distance). We monitor most of the process visually and by reporting the RMSE of the registration at each step. The average RMSE of all the registered scans is generally on the order of 1 cm. We also reconstruct a 3D model from the recovered ground-truth trajectory and the camera scans to verify that the output is similar to the LiDAR model.

From each pair of SLAM trajectory and recovered ground-truth trajectory, we provide statistics on the Absolute Trajectory Error (ATE), which measures the difference between each SLAM pose and its corresponding reference pose, and the Relative Pose Error (RPE) that quantifies the error on pose deltas (the error over a certain distance; in this case, per one meter).

1.6 Results

1.6.1 Arroyo Tapiado Mud Caves

We performed a controlled set of experiments at the Arroyo Tapiado Mud Caves site near the University of California San Diego. This location was the best option for replicating the archaeological conditions in Guatemala in a local setting where all of the laboratory resources would be available during prototype development. First, we defined an area of approximately 50 meters in length and scanned this area with the same FARO LiDAR system that was used to scan the El Zotz excavations. Then, we ran a series of scans using our SLAM scanning solution with three different sensors: Kinect v1, Kinect v2, Realsense D435, and two different algorithms: ORB-SLAM, RTAB-MAP. All the scans were performed by the same operator, and followed a similar trajectory and scanning pattern. Additionally, we recorded all sensor data, and replayed them through the InfiniTAM v2 algorithm afterward.

Here we present the results of comparing the output data from the SLAM algorithms to the LiDAR 3D model.

Cloud-to-cloud difference

Figure 1.2 shows an example of the cloud-to-cloud difference between a SLAM scan and a LiDAR scan. On top is the LiDAR scan of the mud caves site where we have focused our experiments. In the middle is the point cloud obtained from scanning the cave with the Kinect v2 and the RTAB-MAP algorithm. First, we aligned these models globally using ICP. Then, for each point in the SLAM point cloud, we computed the distance to the nearest neighbor in the reference LiDAR scan. The bottom model shows these point-to- point distances colored on a color scale from white (small distance) to red (large distance). In general, we notice more errors accumulated where the cave presents a turn. This is mainly due to two reasons. First, more drift is accumulated in areas with less visibility because the sensor has a more limited view of environment features. The other reason is that turns are places where sensor noise tends to

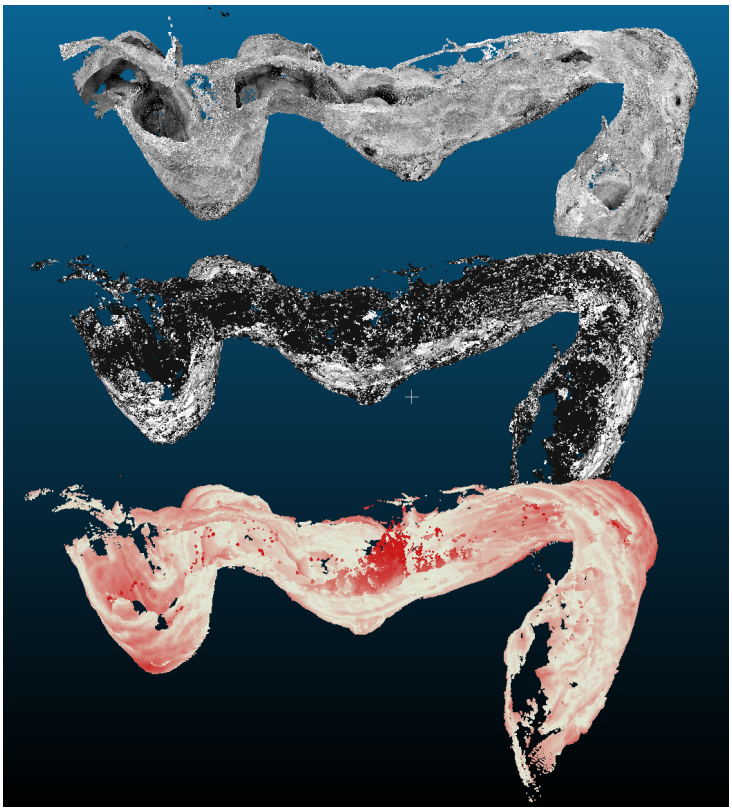


Figure 1.2. From top to bottom: The LiDAR scan of the mud cave, the SLAM scan using RTAB-MAP and Kinect v2, and the same SLAM scan with point-to-point error with the LiDAR scan highlighted (darker shade of red = larger error).

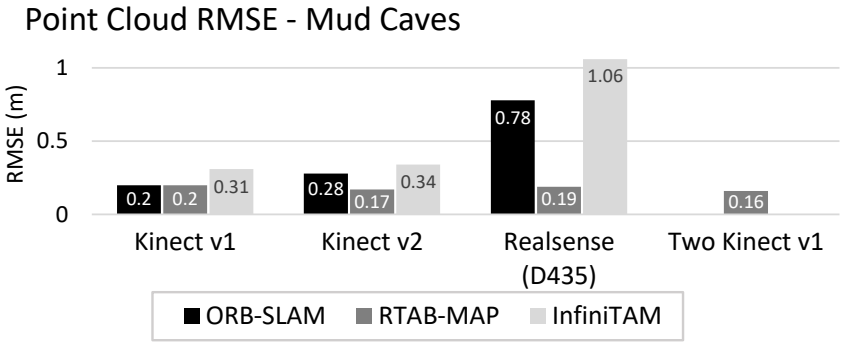


Figure 1.3. RMSE of the cloud-to-cloud difference between SLAM algorithm results and the LiDAR cloud, for the mud caves site.

Table 1.2. Absolute Trajectory Error (APE) and Relative Position Error (RPE) on different combinations of sensors and algorithms. *ATE* compares the trajectories aligned on the first pose only. *ATE (aligned)* compares trajectories globally aligned. *RPE* measures the relative error per 1 meter segments. All values are expressed in meters and averaged over the entire trajectory.

	ORB-SLAM		**RTAB-MAP**		**InfiniTAM v2**	
	ATE (aligned)	RPE	ATE (aligned)	RPE	ATE (aligned)	RPE
Kinect v1	0.64 (0.26)	0.04	0.98 (0.33)	0.05	2.10 (0.59)	0.05
Kinect v2	1.08 (0.36)	0.08	0.70 (0.31)	0.06	1.21 (0.56)	0.10
Realsense (D435)	5.24 (2.20)	0.25	1.49 (0.32)	0.11	-	-
Two Kinect v1	-	-	0.87 (0.25)	0.07	-	-

accumulate. The depth quality of RGB-D cameras decreases with the distance, therefore distant obstacles can accumulate the noise from multiple scans.

We summarize the cloud-to-cloud differences for each combination of algorithm and sensor by computing the RMSE over the entire point cloud. The results are presented in Figure 1.3. We can observe a similar error in multiple models using Kinect v1 and v2 on ORB-SLAM and RTAB-MAP. InfiniTAM v2 tends to give a larger error, as expected from an algorithm without loop closure detection. The Realsense camera tends to show a less consistent, larger error. This is due to the technology employed to generate depth data. Stereo matching can be noisier, especially as distance from the sensor increases. Intel provides a number of software filtering options for this camera, and we can expect better results when fine-tuning the process for this specific application. We also compare the results to a setup using two Kinect v1 cameras with RTAB-MAP. The model in this case presents a better RMSE as the software can utilize more sensor data to refine its trajectory calculation.

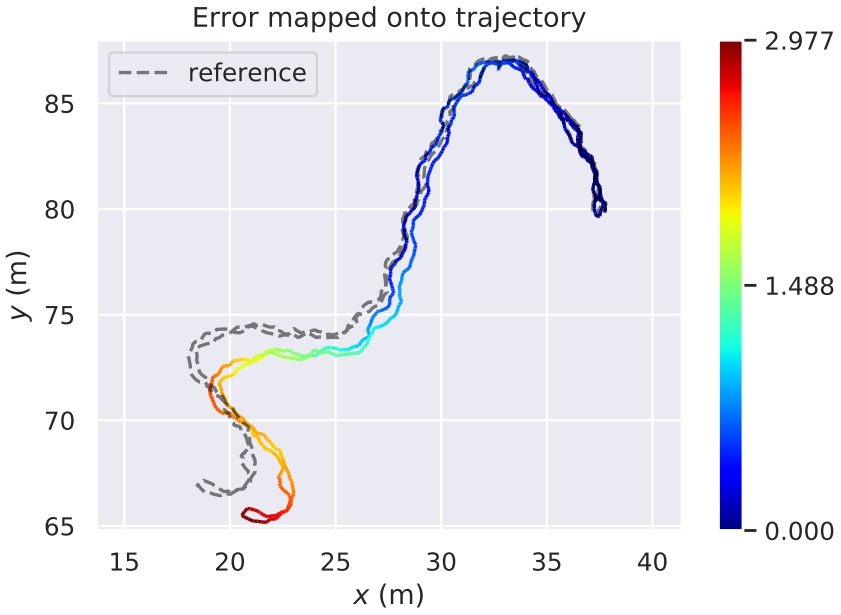


Figure 1.4. Trajectory comparison between RTAB-MAP with Kinect v1 and the recovered ground truth. The position error is plotted on the SLAM trajectory in meters.

Trajectory difference

For each trajectory output from the SLAM algorithms, we recovered the corresponding ground-truth trajectory using the LiDAR data. We measured the Absolute Trajectory Error (ATE) between the two outputs in two different ways: 1) by aligning the first pose of each trajectorythis gives an idea of how much error the algorithm accumulates with respect to the starting position, and 2) by aligning the whole trajectory to minimize the average errorthis informs about the average error over the entire scan.

Figure 1.4 shows an example of the ATE between a trajectory from RTAB-MAP with Kinect v1 and the reference output, with respect to the starting position. We summarize the results in Table 1.2. The InfiniTAM results with the Realsense camera are not included, as they are too noisy to provide a good ground truth reconstruction. Generally, the results from the Kinect cameras are similar. Despite possessing a global shutter camera, the Kinect v2 does not show a consistent improvement, however it is possible to increase the resolution to improve the results. The Realsense camera creates more noise, and this translates to a less accurate trajectory.

Figure 1.5. Comparison of trajectories with and without loop closure using RTAB-MAP with Kinect v1.

The InfiniTAM algorithm accumulates a lot of error over time as its tracking is based on depth data only. This is clearly reflected by a much worse ATE (large global error), but still keeping a similar RPE as other algorithms (good local consistency).

Loop Closure Analysis

While loop closure is a crucial step of most SLAM algorithms in order to increase the accuracy and reduce the drift, loops are very rare in tunnels and excavations. In this type of environment, one possible way to trigger a loop closure is to turn around, travel back to a previous point in the trajectory, and turn around again. This last turn around allows the algorithm to recognize a visited place by matching similar visual features. We analyze the difference between closing the loop and not closing it. We run the sensor data through the ORB-SLAM algorithm, once keeping the loop closure, and once stopping before backtracking. Figure 1.5 shows the profiles of the trajectories with and without loop closure, and the recovered ground truth. Both SLAM trajectories present a small drift over time with respect to the ground truth, but the drift is clearly higher without loop closure.

1.6.2 El Zotz Archaeological Site

We tested our scanning setup in a cultural heritage context at the archaeological site of El Zotz, Guatemala, and report our results here. Due to various constraints of the field, not all of our experiments were executed in the exact same conditions. For this reason, we replayed the saved camera data through the different algorithms to increase the consistency. We collected data with the Kinect v1 and v2 cameras. Due to algorithm failure, the Kinect v2 data do not perform any loop closure. We also collected data with a Realsense ZR300 camera, however the results are too noisy to be properly analyzed here.

We also compared the SLAM models to a LiDAR scan taken at a previous time, and therefore the models can differ at certain specific areas since excavation conditions changed as archaeologists expanded new tunnels and backfilled other branches. The cloud-to-cloud difference is higher for this reason, and can only be used as a comparison metric between SLAM algorithms and sensors. The trajectory reconstruction has been manually adjusted to ignore the areas with differences.

Cloud-to-cloud difference

Figure 1.6 shows an error plot of a SLAM model against the LiDAR model. The areas of high errors mostly correspond to the differences between the scans. Otherwise we notice a similar trend with the mud caves models, where certain walls accumulate noise from sensor. This is more noticeable with ORB-SLAM as the sensor data are simply merged together without any kind of filtering.

Figure 1.7 shows the summary of RMSE for the two cameras. ORB-SLAM with Kinect v2 fails to reconstruct, but otherwise presents better results for Kinect v1. In this case, the Kinect v2 gives better results even though there is no loop closure. This environment is more difficult and scans are very shaky, which can be handled better by the global shutter camera.

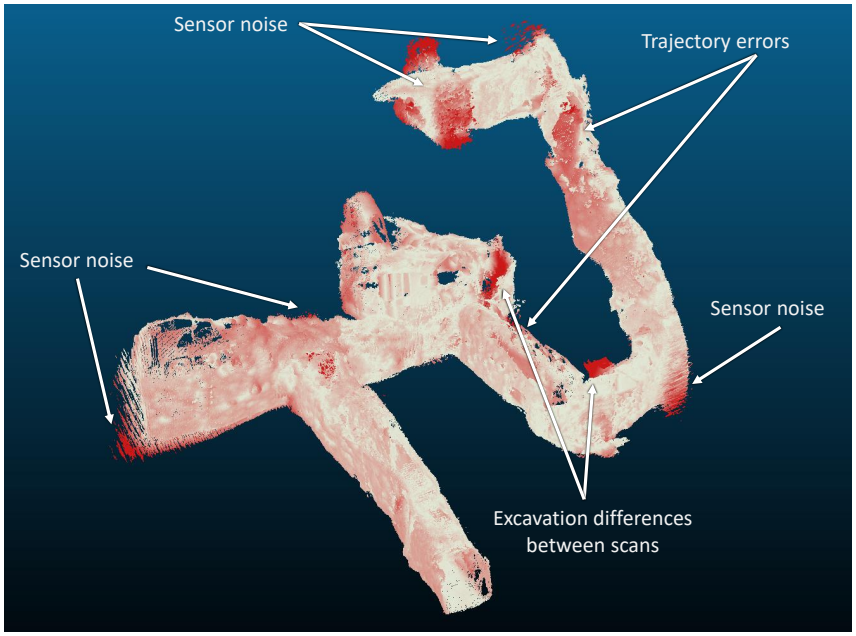


Figure 1.6. Point-to-point difference between the point cloud from ORB-SLAM with Kinect v1 and the LiDAR point cloud. We show the SLAM cloud with the difference colored on each point. Darker red corresponds to a larger error, which happens in areas with concentrated sensor noise, and areas where the two models differ in geometry.

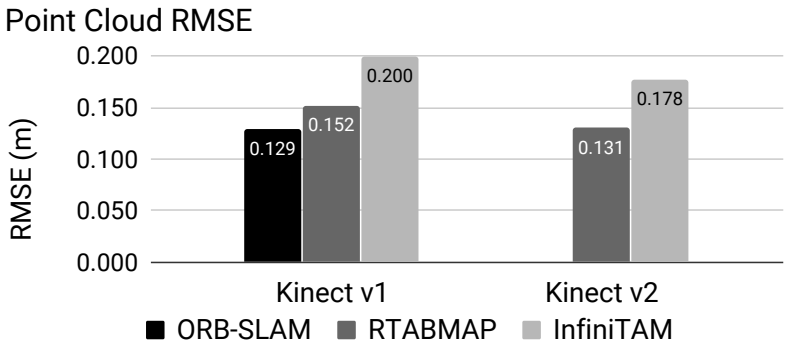
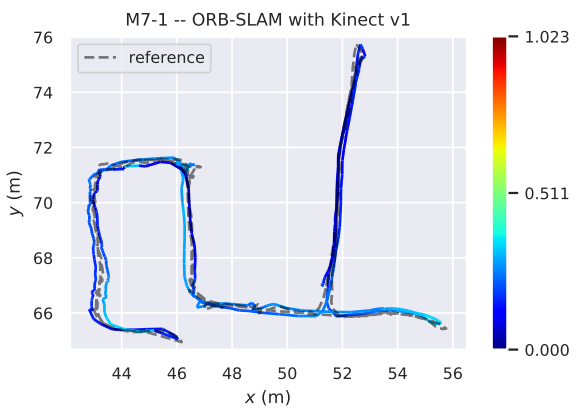
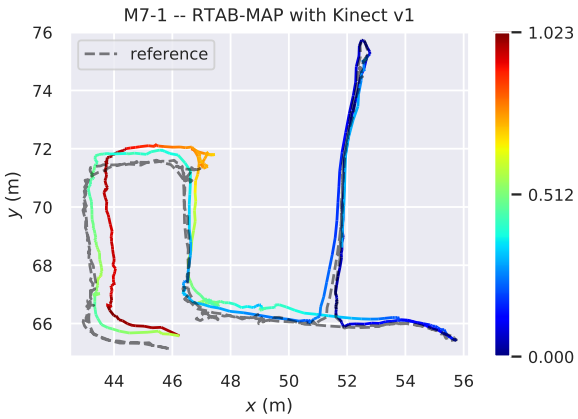


Figure 1.7. RMSE of the cloud-to-cloud difference between SLAM algorithm results and the LiDAR cloud, for the M7-1 excavation site.



(a) Trajectory error using ORB-SLAM with Kinect v1.



(b) Trajectory error using RTAB-MAP with Kinect v1.

Figure 1.8. Top view of the trajectory comparison between SLAM results and the recovered ground truth. The position error is plotted on the SLAM trajectory in meters.

Table 1.3. ATE and RPE in meters, on the M7-1 excavation.

	ORB-SLAM		**RTAB-MAP**		**InfiniTAM v2**	
	ATE (aligned)	RPE	ATE (aligned)	RPE	ATE (aligned)	RPE
Kinect v1	0.21 (0.19)	3.32*	0.47 (0.29)	0.06	2.27 (2.49)	0.08
Kinect v2	-	-	0.93 (0.17)	0.08	1.10 (0.47)	0.10

* The trajectory has a large discontinuity causing a very large value for RPE.

Trajectory difference

We recovered a ground-truth trajectory for our camera scans based on the LiDAR data. Figure 1.8 shows the trajectory errors from ORB-SLAM and RTAB-MAP with Kinect v1. In this case, the RTAB-MAP algorithm has accumulated more drift than ORB-SLAM. We summarize the trajectory metrics in Table 1.3. The results are similar to what we measured in the mud caves environment, with InfiniTAM results being worse in global consistency compared to the image-based tracking algorithms.

1.7 Analysis and Discussion

In this Section, we discuss high-level trends and directions to take to maximize the ac- curacy of SLAM-based scanning, based on the results presented in Section 1.6 and our observations in the field.

1.7.1 Sensors

The RBG-D cameras that we used are broadly divided into two categories: active depth sensing (Kinect v1 and v2), and passive depth sensing (Realsense). Both our results and our observations confirm that active sensing produce the best results in terms of 3D scans. The difference in technology for active sensing (structured light with Kinect v1 and time-of-flight with Kinect v2) has a less significant impact on the result, although there tends to be a slight global improvement with the time-of-flight sensor. Stereo-based sensors produce depth maps that tend to be spatially and temporally noisy, which increases the drift in SLAM algorithms, and also increases the frequency of tracking loss during data collection. The ZR300 camera produced results too noisy to be analyzed in Section 1.6.2. However, it is important to note that the active sensors can only operate indoors, and would not be suitable to scan the exteriors of sites. Additionally, the stereo sensors can be improved in software through the fine-tuning of filters, although this increases the complexity of the scanning system.

1.7.2 Algorithms

The algorithms can also be divided in two categories: feature-based tracking (ORB-SLAM and RTAB-MAP), and registration-based tracking (InfiniTAM). Feature-based algorithms mostly rely on image features, and therefore can more easily recover from tracking failure. However, these algorithms cannot recognize the same tunnel viewed from a different point-of-view (e.g., after turning around), and must rely on loop closure to create a consistent model in that case. Registration-based algorithms rely on depth data only, and as such can recognize an

environment based on its geometry only. These algorithms tend to be less stable in terms of tracking as they rely primarily on frame-to-frame alignment. InfiniTAM v3 is a good compromise, but our experiments failed to properly scan on long distances. While InfiniTAM v2 can be used for scanning without much tweaking, InfiniTAM v3 would require more tuning to be useable in these conditions. Feature-based algorithms worked better in our experiments, but they did require particular attention to the lighting of the scene. Our scanning system is equipped with a light panel emitting diffuse light, which minimizes the negative effects on the SLAM algorithm, however a strong, static light is recommended whenever possible.

1.7.3 Scanning Procedure

In Section 1.6, we highlight the importance of creating a loop closure, even when the environment presents no obvious such loop. This procedure applies to feature-based algorithms, or algorithms capable of handling loop closure, but is still useful in all cases simply to add new points-of-view of the scene and increase the number of details. There are also several good practices that we have noted during our experiments. First, each depth sensor has its own specifications in terms of sensing range and field of view. These limitations are very important for the user to keep in mind, as they generally make the difference between a failed scan and a good scan. Typically, there should always be a recognizably distinctive portion of the scene (containing visual features and geometric features) within the nominal range of the sensor (especially above the minimum range). Second, the scanning speed is important. While generally faster than other techniques, SLAM-based scanning is sensitive to fast motion which introduces errors from: motion blur, lens distortion from rolling shutter cameras (minimized by using a global shutter camera such as Kinect v2), more distance between frames (minimized by using higher frequency cameras such as the Realsense D435), or simply breaking the small motion assumption made by registration-based algorithms.

1.8 Conclusion

In this chapter, we have analyzed multiple hardware and software solutions to help with the documentation of cultural heritage within restricted spaces. Through multiple experiments in the field, we have established an accuracy value for different combinations of sensors and algorithms in these low-light conditions. In terms of software, image-based tracking algorithms tend to perform better for large-scale area scanning, but may offer a sparser 3D model without post-processing. Dense SLAM algorithms relying on depth tracking offer a good solution for small areas but do not scale well, unless implemented with loop-closure solutions. In terms of sensors, it is clear that active depth sensing solutions targeted at indoor use perform better than stereo-based cameras in an underground environment. We have found that in average, the best performing combination of sensor and algorithm for our prototype scanning system is a Kinect v2 with RTAB-MAP. More generally, a global shutter camera with a time-of-flight depth sensor, running a feature-based SLAM algorithm, is likely to perform better.

SLAM algorithms provide a low-cost alternative to LiDAR, and a fast alternative to photogrammetry methods. They are generally limited in the quality of results, and we have quantified these limitations for the use-case of documenting archaeological underground excavations. The proposed system is ideal for the rapid documentation of areas with extensive damage from looting where having an expensive instrument in difficult field conditions is impractical. Likewise, such a system has cultural heritage documentation applications in high-risk areas where archaeological sites are damaged or threatened in order to make rapid assessments of impacts and resource requirements.

Acknowledgments

The authors would like to thank the following persons for their help and contribution to this work: Proud Heng, Nathan Hui, Tim Jiang, Waseem Khan, Etsu Nakahara, Giovanni Vindiola, Danbing Zhu.

Chapter 1, in part, has been submitted for publication of the material as it may appear in the Journal of Cultural Heritage Management and Sustainable Development. “Low-Cost 3D Scanning Systems for Cultural Heritage Documentation”. Gautier, Quentin; Garrison, Thomas; Rushton, Ferrill; Bouck, Nicholas; Lo, Eric; Tueller, Peter; Schurgers, Curt; and Kastner, Ryan. The dissertation author was the primary investigator and author of this material.

Chapter 2

FPGA Architectures for Real-Time Dense SLAM

2.1 Introduction

Simultaneous localization and mapping (SLAM) algorithms provide a compelling solution to create a three-dimensional representation of an environment, as well as tracking the position of an agent within this environment. SLAM is a general technique with many fundamental applications beyond 3D scanning; it is useful in the fields of robotics, computer vision, virtual/augmented reality, and many others. Ideally, a SLAM system works in real-time, provides a detailed 3D map, uses minimal power, has a small physical footprint, and is low-cost. But these are often conflicting constraints that create a complex design space.

Early SLAM algorithms dialed back the algorithmic complexity to achieve suitable performance. For example, sparse SLAM algorithms consider a subset of sensor data and only model the environmental information needed for navigation [31]. This was largely in response to available resources, e.g., low-power compute platforms were not available, and sensors provided relatively low bandwidth information. As the efficiency of compute platforms increased, SLAM algorithms added the ability to model more complex environments. It is now possible to produce a real-time detailed 3D model of the world using dense SLAM aka 3D reconstruction algorithms. To work in real-time, these algorithms must process a high volume of information from large amount of sensor data (cameras, depth sensors, IMU, LIDAR, etc.). These systems often carefully

leverage hardware acceleration techniques, e.g., by operating on GPUs and FPGAs [43, 118, 58].

In many applications, including robotics and cultural heritage documentation, the power consumption of a SLAM system is critical. In the case of robotics, it is common for an agent to provide its own power source (e.g., quadcopter drone). In the case of cultural heritage documentation, many archaeological sites are located in remote areas with limited access to power sources. In these cases, a SLAM system designer must implement a system that minimizes power consumption. One solution is to run the algorithms on an FPGA hardware instead of a general-purpose CPU or GPU. However, FPGAs are notoriously difficult to program, and one must carefully design the hardware to reach the best tradeoffs between throughput and on-chip area utilization.

Our ultimate goal is to determine the best way to implement dense SLAM using an FPGA-accelerated system. This requires us to perform architectural optimizations to carefully balance between resource usage, performance, and accuracy.

First, we propose a straightforward optimization of a dense SLAM application (Kinect Fusion) on a large FPGA board by iteratively improving the architecture to reduce the running time, and reach a single final architecture for one main component of the algorithm. Then, we analyze a second application (InfiniTAM) that implements a different internal data structure, leading to an increased flexibility in the possible optimizations on all the components. In this case, the space of possible optimizations becomes very large, and very slow to explore. Compiling and testing a single design can take multiple hours. As a result, we want to increase our understanding of the effects of each optimization in an effort to improve the design space exploration of similar types of applications.

In the context of the InfiniTAM algorithm, we develop a set of highly parameterized architectures for each of the dense SLAM components (tracking, depth fusion, and ray casting). Each component is outfitted with multiple optimizations that can be tuned to offer tradeoffs between FPGA resource utilization, throughput, and quality of result. We compile thousands of unique designs based on these optimizations, and run each of them on two representative

FPGA platforms. The first is the Terasic DE1 FPGA System-on-Chip (SoC), which is a low-cost, low-power integrated system with an ARM processor and a small FPGA. The second is the Terasic DE5 PCIe board, which has an FPGA that is approximately 7× larger than the FPGA on the DE1. We provide an analysis of the resulting spaces to better understand the complex relationship between the tuning parameters and the output architectures.

We develop the FPGA architectures using the Intel OpenCL SDK for FPGA, which provides the flexibility to perform high-level design tradeoffs and eases the integration into the existing dense SLAM frameworks. We develop a complete system capable of performing real-time dense SLAM (Fig. 2.1) which is guided by our design space analysis. Our designs, design space data, and analysis are made open-source [2] to facilitate follow-on work related to SLAM, FPGA design, and hardware design space exploration.

The main contributions in this work are to provide:

- Highly optimized OpenCL FPGA code for the major components of dense SLAM.
- A new algorithm combining two major components of dense SLAM.
- A comprehensive parameterization of these implementations to easily target different FPGA resources constraints and applications demands.
- A statistical analysis of the design space considering more than 2500 possible implementations with different resource utilization and performance.

In Section 2.2 we discuss related work on FPGA-accelerated SLAM. In Section 2.3, we introduce the 3D Reconstruction framework and propose an architecture for the Kinect Fusion algorithm on FPGA. In Section 2.4, we detail our multiple architectures of the InfiniTAM algorithm. Section 2.5 shows the results of our design space exploration of InfiniTAM, and we conclude in Section 2.6.

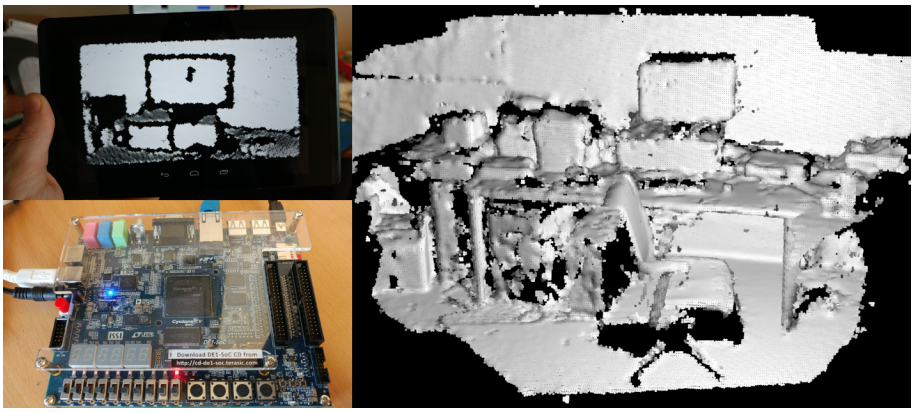


Figure 2.1. 3D model reconstructed in real-time on a DE1 FPGA SoC with the data transmitted from a Google Tango tablet.

2.2 Related Work

SLAM is commonly deployed in applications that utilize high bandwidth sensors, require real-time results, and have a limited power budget [57]. This has naturally pushed SLAM designers towards hardware accelerated platforms.

FPGAs are a particularly attractive platform due to their power efficiency. For example, the bottleneck in visual SLAM algorithms (i.e., using cameras as sensors) is feature detection (detecting points of interest in an image) and feature extraction (encoding the visual features for distance calculation). Ulusel et al. [111] analyze one feature detection algorithm (FAST) and two feature extraction algorithms (BRIEF, BRISK) on embedded CPU, GPU, and FPGA. Feature detection and extraction are common tasks in SLAM. Their results show that the FPGA implementation outperforms the CPU and GPU in both power and performance.

The complexity of SLAM algorithms makes it difficult to implement an entire end-to-end system utilizing solely an FPGA. FPGA SoCs are an appealing option as the algorithm can be split across hardware and software. For example, Tertei and Devy [107] implement a version of SLAM based upon an Extended Kalman Filter. They perform matrix multiplication on the FPGA and the remainder of the algorithm in software. Nikolic et al. [88] build a visual-inertial motion estimation system. They offload the feature detection (Harris corners / FAST corners)

onto the FPGA. Similarly, Aguilar-Gozalez et al. [6] describe an FPGA implementation of the detection/extraction process to increase the number of features detected by standard feature detection algorithms.

Other works utilize the FPGA for a larger portion of the application. The authors in [103] implement a particle filter SLAM on the FPGA. The input comes from a sparse laser scanner and the map is created as a simple occupancy grid. Another work implements a large portion of the Scan-Matching Genetic SLAM (SMG-SLAM) algorithm [79] on an FPGA. SMG-SLAM is similar to our algorithm, but takes its input from a sparse laser range finder. The result is stored in an occupancy grid map with a resolution between 2 cm and 12 cm and a fairly low number of grid cells (up to 724). While occupancy grids are an efficient map representation for navigation purposes, other representations such as Signed Distance Function [29] are more adapted to dense 3D reconstruction. More recently, Boikos and Bougnais [13, 14] accelerate the *semi-dense* LSD-SLAM algorithm on an FPGA SoC achieving 22 frames per second on a 320x240 input visual frame. We are able to handle denser environmental maps than these projects.

2.3 FPGA Architectures for Kinect Fusion

Dense SLAM algorithms which focus on creating detailed 3D models are often classified as *real-time 3D reconstruction* algorithms. We call 3D reconstruction any algorithm whose main purpose is to create a realistic 3D model of an environment. Kinect Fusion [86] is a notorious work in the domain of real-time 3D reconstruction based on low-cost RGB-D (RGB + Depth) camera sensor.

Our first investigation to port dense 3D reconstruction on FPGA is based on Kinect Fusion. We present the main components of this application, which have been reused and extended by many other works since Kinect Fusion, such as the InfiniTAM algorithm that we use for our second set of experiments (Section 2.4). In this section, we also describe our partial FPGA

implementation of this algorithm, the results and conclusions that we obtained by running the application on multiple hardwares.

2.3.1 Dense SLAM Overview

This work is based on Kinfu, an open-source implementation of Kinect Fusion that exists within the PCL library [97]. Kinfu runs its major components on a GPU by using the CUDA programming language. Fig. 2.2 illustrates the 3D Reconstruction framework used in Kinfu. It takes as input a depth map from an RGB-D camera (e.g., the Microsoft Kinect) and processes this depth map through three main modules:

- **Tracking** estimates the camera's position in 3D space based upon the current depth map and previous data.
- **Depth Fusion** (also called Volumetric Integration) integrates the input depth map into the current 3D model.
- **Ray Casting** fetches a 2D view of the model (Voxel Map) at the current camera pose.

This is an iterative algorithm where the three modules run every frame, and thus they must run quickly if real-time behavior is required. Below we give a brief description of the major components of the system, with more details in the original paper.

Map Representation

The 3D model is stored in a grid of voxels using the Truncated Signed Distance Function (TSDF) [29]. TSDF represents the world as a set of voxels. Each voxel records the distance to the nearest surface, along with a weight value to encode a confidence. The distance is normalized to a maximum value called *truncation distance*. The PCL implementation of Kinect Fusion instantiates a 3D grid of voxels of size $512 \times 512 \times 512$ that represents a real space of size $3m \times 3m \times 3m$. The grid is created at the initialization of the algorithm and cannot move; as a result, the model must be contained within this area. Additionally, all the voxels are instantiated,

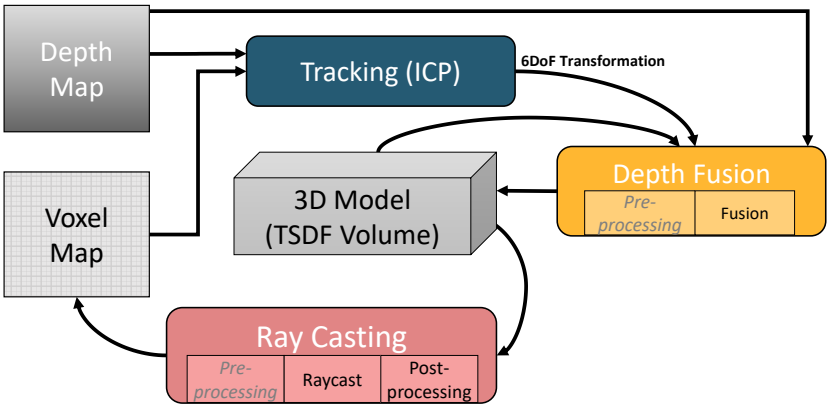


Figure 2.2. 3D Reconstruction workflow. Each iteration of the algorithm takes the depth map from the RGB-D camera as input, then goes through three steps: ICP, Depth Fusion, and Ray Casting. The overall process is the same on Kinect Fusion and InfiniTAM. The pre-processing steps only exist in InfiniTAM.

regardless of whether they have received sensor information or not. They are organized into a plain 3D grid, accessible by a regular set of (x,y,z) coordinates.

Depth Fusion

The depth fusion algorithm iterates over all the voxels in the 3D grid. Each voxel is projected into the depth map, the distance to the camera center is compared to the depth map value, and if the two distances are within the truncation distance, the voxel's current distance to the surface is merged with the old distance value using a weighted average.

Ray Casting

The ray casting algorithm consists of sending a ray from each pixel of a 2D view into the 3D TSDF model, to find a zero-crossing (a point where the distance to the nearest surface varies from positive to negative). The resulting distance is saved as a 3D point in a Voxel Map. A post-processing step estimates a surface normal for each resulting 3D point, and optionally calculates a color intensity on each point for display purposes.

Tracking

Tracking is based on 3D registration with the Iterative Closest Point (ICP) algorithm [9]. ICP finds the optimal alignment between the input depth map and the 2D projection of the current model from the last camera position (the output of Ray Casting). The calculated transformation between these two inputs represents the motion between the previous and the current frame. The details of this particular implementation of ICP are presented in the original Kinect Fusion paper [86].

2.3.2 Iterative Closest Point (ICP) on FPGA

The particular version of the Iterative Closest Point algorithm that is used in Kinfu is presented in [56, 86]. We also provide an algorithm listing to explain the process (Algorithm 1). Note that this listing is based on the InfiniTAM implementation of ICP (Section 2.4), but it is functionally equivalent. The main difference is that the original GPU implementation in Kinfu is divided into two kernels, the second one taking care of summing all values together (line 13).

The ICP algorithm aligns the current depth map with the Voxel Map calculated in the previous Ray Casting iteration. It finds pairs of corresponding points by 1) projecting points from the previous and the current voxel maps into a common 2D frame (lines 5 to 8), and 2) comparing the distance and angle between pairs against threshold values to determine if they are inliers or outliers (line 10). The distance and angle between corresponding points are turned into a system of equations represented by two triangular matrices H and ∇. These matrices are summed together over all points, and are solved on the CPU to update the global transformation matrix between the frames. The initial transformation matrix is set to the transformation of the previous frame, then updated at each iteration. ICP operates on multiple resolutions of the input data (original resolution, half resolution on each axis, and a quarter resolution on each axis).

We first ported the original CUDA implementations of ICP to OpenCL by keeping the same structures and features, with minimal differences. We consider that the OpenCL version is

Algorithm 1: ICP Algorithm

Input : Depth map; Voxel Map; Normal Map

Initialize H_g and ∇_g to 0
for *each pixel (x,y)* **do**
    Fetch depth D at (x, y)
    if $D = 0$ **then** continue ;
    Transform D with current estimated pose $\rightarrow p_{cur}$
    Project p_{cur} into 2D view $\rightarrow (i, j)$
    if (i, j) *not valid* **then** continue ;
    Fetch 3D point from Voxel Map at $(i, j) \rightarrow p_{prev}$
    if p_{prev} *not valid* **then** continue ;
    if $distance(p_{prev}, p_{cur}) > threshold$ **then** continue ;
    Fetch 3D normal from Normal Map at $(i, j) \rightarrow n_{prev}$
    Calculate H_l and ∇_l
    Accumulate H_l and ∇_l into H_g and ∇_g
end
Save H_g and ∇_g into global memory

Output : H_g and ∇_g

the *baseline implementation*. Then, we made incremental modifications to better optimize the algorithm on FPGA. We present the different steps that we took to optimize the ICP algorithm.

Baseline implementation

The original GPU code for ICP is divided into the *search kernel* that calculates the correspondence between every points, and the *reduction kernel* that sums the transformation of all the corresponding points. The summation is done with a tree reduction that uses shared memory and synchronization between compute units.

Kernel specialization

We first moved the entire tree reduction to the second kernel (*specialize kernels* in Figure 2.3). This removed a lot of thread synchronization in the search kernel and resulted in a better overall performance. However the data had to be transferred through global memory, which became a bottleneck due to large bandwidth requirements.

Loop unrolling and index dependency

The original code used a double nested loop, where one index was function of the other. This resulted in poor optimization from the compiler. We removed the index dependencies and unrolled the loops manually (as it used slightly less board space than using OpenCL *pragma unroll*). Figure 2.3 presents the improvement of both modifications separately.

Tune reduction parameter

In the original GPU-style tree reduction, 512 work-items ran an individual summation of their own data, then a tree reduction was performed in local memory to get the final sum. We reduced the number of work-items to a number of 64 obtained experimentally to get much better performance. However we ultimately replaced the entire tree reduction as seen below.

Altera channels and shift registers

The GPU-style reduction was very inefficient on a FPGA, mainly because of the large data transfer through global memory. The access time and bandwidth limitation of this memory was slowing down both kernels. To remove this unnecessary access, we used Altera OpenCL *channels*. Channels are created in hardware as simple FIFO queues that can transfer data between kernels in 1 cycle per value. We used 27 of them to transfer the transformation vectors of 27 values to the reduction kernel. We tried to implement 27 independent reduction kernels, but found out that a single one was performing much better. We implemented it as an OpenCL task (single execution unit) to take advantage of loop pipelining provided by the compiler. To remove the inherent data dependency of a summation, we used a shift register. Shift registers have to be implemented manually by looping through the data, but this structure is recognized by the compiler and implemented efficiently in hardware. We used a 32 elements-wide shift register, and we chose the depth to be 8 values based on performance estimations. These techniques decreased the running time by 55%.

Other optimizations

a) We used compiler flags to optimize floating-point operations by reordering. b) We simplified the indexing of input arrays for the compiler to perform more aggressive optimizations. c) We changed the layout of the input data from Structure of Arrays to Array of Structures to take advantage of a 512-bits wide global memory access. d) We experimented with fixed-point arithmetic, however there is no native support in OpenCL for this format and this resulted in suboptimal design that did not fit on the FPGA board. Also, this implementation of ICP performs operations on both large and small numbers. This made it impractical to choose appropriate precisions for the integer and decimal parts without losing overall accuracy. e) We could not use SIMD operations or duplicated compute units because Altera channels cannot be vectorized, and because the ICP algorithm was using too much area on the board.

2.3.3 Depth Fusion and Ray Casting

Depth Fusion and Ray Casting are algorithms that interact with the 3D model to either retrieve or store information. Depth Fusion merges data from the depth map into the 3D model, and Ray Casting fetches 3D points from this same model. The 3D model is represented by a TSDF volume, and implemented with a grid of $512 \times 512 \times 512$ voxels. This large data structure is an issue on FPGA as it must stay in the external memory, and the total bandwidth between the FPGA and the memory is limited. Data caching is generally not an option without completely changing the behavior of both algorithms. Depth Fusion must step through every single voxel, which creates a large memory bottleneck. Ray Casting does not access all the voxels, however the access pattern is very irregular and difficult to cache without a dedicated pre-processing step.

One solution to mitigate this issue is to reduce the size of the grid. However, this solution would severely limit either the voxel resolution or the map size. The data structure would need to be reorganized for an efficient access by a different hardware than a GPU. For these reasons, we only implemented a baseline version of Depth Fusion with basic optimizations, and did not

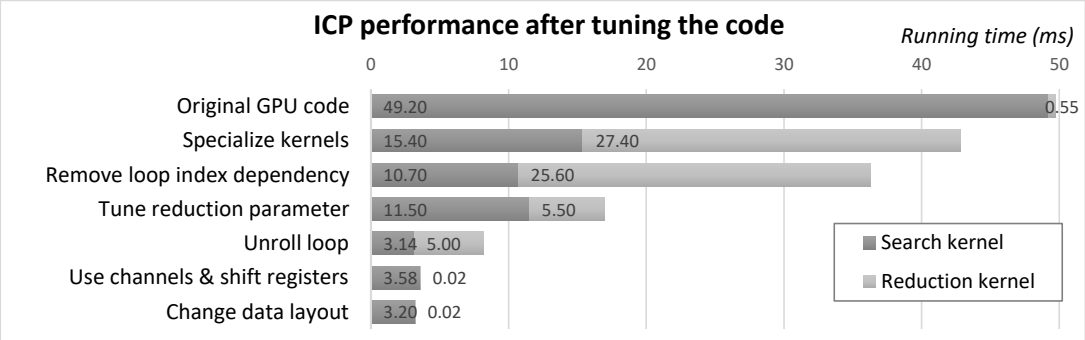


Figure 2.3. Running time of 1 iteration of ICP at full resolution for the major modifications made to the baseline GPU implementation. The modifications are cumulative. The search kernel is the part of the algorithm that compares every vertex to find corresponding points. The reduction kernel sums the equation matrices for all the points.

investigate the Ray Casting algorithm further in the context of this application.

2.3.4 Kinect Fusion Running Time

First we measured the performance of each algorithm running on its own, using a set of fixed input data. Then we integrated the tuned implementations into the complete Kinfu application. We ran the original and modified application on a desktop computer with an Intel i7-4960X CPU, a NVidia GTX 760 GPU, and a Terasic DE5 board with an Altera Stratix V FPGA.

ICP

We measured the running time of ICP for one iteration at full resolution. The baseline FPGA implementation takes 49.9 ms on a FPGA. After tuning the code, we decreased the time to 3.22 ms on the FPGA with a clock frequency of 197 MHz. The running time of intermediate versions of the code are summarized in Fig. 2.3. However we could not achieve one cycle per element because of several factors, including reads from unpredictable locations in global memory. We reduced the impact of these reads by grouping the elements, reducing the number of memory accesses, but it was not possible to make use of local memory since there is no data reuse. The execution pipeline is also delayed by multiple floating-point computations with

dependencies, including divisions and square root functions. Another big limitation is the area utilization. Because ICP uses around 78% of the board logic, we could not integrate another algorithm on the same hardware.

Depth Fusion

Our basic implementation of Depth Fusion runs on the FPGA in 100 ms, which is very slow compared to the rest of the application. The algorithm performs operations on 512 MB of data and there is little data reuse to optimize memory access. Transferring the data back and forth from the FPGA to the GPU would take too long and would require too much bandwidth (15 GB per second) to justify any peformance increases from the Altera compiler and the FPGA. The baseline version of volumetric integration used 89% of the board logic, which precluded placing the VI kernel on the same board as the ICP kernel.

Integration with Kinfu

We ran the Kinfu application on the GPU and FPGA described above. Due to the area limitation discussed before, we could only run one algorithm on the FPGA (Depth Fusion or ICP) and the rest of the project on the GPU. For the Depth Fusion algorithm, we realized that the data transfers between the GPU and the FPGA would take a significant portion of time, because the amount of data copied back to and from the FPGA totals over 512 MB. Given that, achieving a real-time solution for the resolution of 512x512x512 could not be accomplished in real-time. For ICP, we had to copy the voxel and normal maps of two frames from the GPU memory to the FPGA off-chip memory. Even with aligned memory buffer on the host, this took around 18 ms and the entire ICP process 37.8 ms. The entire project ran at 13.5 FPS in average. To get a real-time performance, we removed the high-resolution input data and ran 12 iterations at 320x240 and 8 iterations at 160x120. The memory transfer took around 5 ms and the ICP algorithm ran in 13.5 ms with no noticeable accuracy loss. The entire system (ICP on FPGA, everything else on GPU) was running at 26-28 FPS.

2.4 FPGA Architectures for InfiniTAM

We create FPGA architectures for the SLAM framework called *InfiniTAM* [58] which is itself derived from Kinect Fusion, and runs on a multicore CPU or on a GPU

2.4.1 Overview

InfiniTAM follows the same process as Kinect Fusion (Fig. 2.2), but the main difference is the utilization of a hash table to store the TSDF representation of the 3D model. A hash table is an efficient way of storing a TSDF volume by only keeping non-empty voxels. Moreover, hash tables are shown to perform very efficiently on FPGAs [55, 110] in terms of memory accesses. InfiniTAM leverages the voxel hashing representation of [87], with a hash function based on the coordinates. The hash table actually references *blocks* of $8 \times 8 \times 8$ voxels, while the actual voxels are stored in a flat memory buffer.

Additionally, InfiniTAM presents a few more differences with respect to Kinect Fusion. The Depth Fusion and Ray Casting algorithms contain a pre-processing step that leverages the hash table to limit the number of voxels used in the actual computation. The ICP algorithm is almost identical with the addition of an option to solve the transformation for rotation only or translation only.

In this section, we describe our proposed FPGA architectures for the main components of InfiniTAM, along with the optimization options that we enable. We focus on the core algorithms, without pre- or post-processing steps. For each algorithm, we develop parameterized OpenCL code that is synthesized to an FPGA using the Intel FPGA OpenCL SDK. Different optimizations are enabled, disabled, and tuned using parameters, also called *knobs* for design space exploration purposes [74, 77]. We define many knobs (e.g., loop unroll factor, memory layout, etc.) that generate designs with different performance, resource usage, and accuracy. In the following, we describe the OpenCL implementations for each algorithm, with a focus on the key optimization knobs that we analyze in Section 2.5.

We also propose a new algorithm that combines Depth Fusion and Ray Casting to improve throughput. We refer to this algorithm as the *Combined Kernel*.

2.4.2 Depth Fusion

The Depth Fusion algorithm takes as input the depth map, a list of hash table entries indexes, the hash table, and the buffer containing actual voxel data. The algorithm contains an outer loop to process each hash table entry (block), and an inner loop to process the 512 voxels inside each block. In the following, we describe how we modified this implementation with FPGA-specific optimizations.

Loop Optimizations

The OpenCL compiler offers multiple options to implement loops. This includes using OpenCL *work-items* (WI) (on FPGA, a WI is generally interpreted as a single – data-dependency free – stage of a pipelined loop), a simple *for* loop that automatically gets pipelined, or *work-groups* (groups of WI) that enable coarse-grained parallelism. We provide knobs to switch between these representations. The coarse-grain parallelism enabled with work-groups is an OpenCL feature called *compute units*. Each compute unit is a duplication of the kernel to increase the parallelism and thus easily scale the task on larger devices. We provide another knob to control the number of compute units.

Memory Optimizations

Input buffers can be cached using *local memory*, which has better latency for non-predictable accesses, such as depth map indexing. We implement a knob to optionally pre-load the depth map into local memory in a predictable way. In this case, the depth size must be fixed (another knob) and the design uses more BRAMs. Lastly, to prevent a potential memory bottleneck when accessing voxels, we implement a knob to cache one block of voxels in local memory for reading and optionally for writing.

Additional Optimizations

We also provide knobs to control the unrolling factors on various loops. Another knob controls the placement of a branch condition, which either groups voxel accesses together or keeps an early branch to potentially terminate the loop earlier.

2.4.3 Ray Casting

The Ray Casting algorithm iterates over the pixels of a 2D view to project 3D information into that view. For each pixel, it steps along a 3D ray and reads the distance value from the hash table until that distance becomes negative. Then the algorithm refines the location of the surface by stepping backwards along the ray using interpolated distance values. Estimations of the starting and ending points of the ray are pre-calculated in a downsampled *min/max* input map. The output is a *Voxel Map*, an organized point cloud containing one 3D point per pixel.

Memory Optimizations

The data access pattern is complex and non-contiguous particularly when fetching voxel data. Consecutive accesses to the same data can be manually cached, and the compiler also provides an automatic cache mechanism. We provide knobs to enable/disable these caches, which provides tradeoffs between BRAM usage and cache speedup. The interpolation fetches 8 different values for averaging. We provide the option to disable this interpolation, which may decrease the overall quality of the SLAM system. Finally, the algorithm ends with a *Refine* step that creates another voxel data access. We can disable it with a knob. These last two knobs are often necessary to save logic elements and fit the kernel on smaller devices.

Other Optimizations

Just like Depth Fusion, we can also switch between a *for* loop and work-items for the outer loop, and we have the option to fix the 2D image size to simplify some calculations. We can also remove some coordinate system transformations and memory accesses by simplifying

Algorithm 2: Combined Algorithm

Input : Depth Map; Visible blocks IDs; Hash table; Voxel Buffer; Truncation threshold μ

```
for each visible block ID do
    Fetch block B from hash table
    From B, fetch pointer to voxel block in voxel buffer
    for each voxel V = (x,y,z) in the voxel block do
        (g_x, g_y, g_z) ← Calculate global coordinates of V
        (c_x, c_y, c_z) ← (g_x, g_y, g_z) to camera view
        (i, j) ← Project (c_x, c_y, c_z) into 2D image
        D ← Get depth at (i, j)
        (w, d) ← Fetch weight and distance from V
        if w ⩽ w_max or |c_z − D| ⩽ μ then
            (w', d') ← New weight and distance
            Store new voxel V ← (w', d')
        end
        d ← Fetch distance from V
        if |d| < ProjectionMap[i,j] then
            c_z ← c_z + d
            (x, y, z) ← (c_x, c_y, c_z) to global coord.
            Save (x, y, x) position into Voxel Map
            ProjectionMap[i,j] ← |d|
        end
    end
end
```

the start and end points of the ray. The ray can start at a depth of 0, and end at the maximum sensor range, which makes an assumption about the input data. We keep both options as knobs.

2.4.4 Combining Depth Fusion and Ray Casting

Ray Casting provides a view of the fused 3D model at the current position, and as such, accesses most of the voxels that have just been updated by Depth Fusion. Thus, it is beneficial to combine both steps into one coordinated process. However these two steps process data in opposite directions. Depth Fusion projects 3D voxels into a 2D view, while Ray Casting sends rays from the 2D view into the 3D volume. Depth Fusion is more efficient as it involves less memory accesses due to the pre-processing step. We create a *Combined* kernel by integrating

Ray Casting into Depth Fusion. We implement the uninterpolated and unrefined version of Ray Casting to avoid significant random accesses to the hash table and the voxel data. The main issue arises when Depth Fusion projects multiple points on the same pixel. Because we lose the sequential aspect of the Ray Casting algorithm, we need a way to select which distance (d) to keep on each pixel.

The Combined algorithm is presented in Algorithm 2. There are two computation blocks: the Depth Fusion block at line 10, and the Ray Casting block at line 15. The Depth Fusion is similar to the orginal implementation. The Ray Casting block writes the distance d of the current voxel to a *projection map* which is the same size as the 2D image view. The goal of the projection map is to save only the smallest d in the projected pixel. When a new distance d_{new} is projected into the map, it is saved only if $|d_{new}| < |d|$, at which point the Voxel Map is updated with the new point. This creates a dependency between iterations that is difficult to avoid without duplicating the entire projection map.

Optimizations

This implementation mostly uses the same knobs as Depth Fusion with some restrictions. We restrict the outer and inner loops to be implemented as *for* loops only. We add the possibility of implementing the inner loop as work-items, which in the OpenCL model creates a race condition on the projection map. On FPGA, this race condition occurs depending on the depth of the pipeline, which can create a small loss of precision in the result. We also add a knob to use either local or external memory for the projection map.

2.4.5 Iterative Closest Point (ICP)

ICP aligns the current depth map with the Voxel Map from Ray Casting. It finds pairs of corresponding points by projecting them into a common 2D frame and rejecting candidate pairs based on threshold values. The distance and angle between pairs of points are turned into a system of equations represented by two triangular matrices H and ∇. These matrices are summed

together over all points and solved on the CPU to update a global transformation matrix. ICP has the option to solve for rotation or translation only, which uses smaller H and ∇ matrices. We propose a number of optimizations for the FPGA implementation.

Interpolation

The 3D points and 3D normals from the depth map and Voxel Map can be read with bilinear interpolation. These interpolations can be disabled with knobs.

Branching

This kernel has a lot of branches and therefore a lot of control logic. It contain branches to differentiate between *long* and *short* iterations, i.e., whether we solve for rotation and translation, or only one component. We implement a knob to disable these branches, and fully calculate the matrices even for short iterations. This helps reduce the resource utilization.

Accumulation

This kernel has a lot of data dependency between each iteration due to the matrices needing to be summed together. We implement multiple shift registers to decrease this dependency. To balance between speed and logic utilization, we provide knobs to enable shift registers and control their size. In the case where we do not use shift registers, we implement knobs which unroll the computation and accumulation of the matrices to decrease the latency.

2.5 Experimental Results And Analysis

In this section, we analyze the effects of the various hardware optimizations to provide insight on how to design architectures for real-time dense SLAM systems. To ground our design process in reality, we focus on two different FPGA platforms – an embedded FPGA SoC and a higher performance PCIe FPGA system.

Table 2.1. Summary of the knobs for all four algorithms, along with their names used in Section 2.5

Depth Fusion		Ray Casting	
Name	Description	Name	Description
ComputeUnits	Num. compute units	**Interpolate**	Enable interpolation
XyzLoop	Inner loop type	**Refine**	Enable refine step
EntryIdLoop	Outer loop type	**IndexCache**	Voxel access cache
EntryIdNumWI	Outer loop num. WI	**HashCLcache**	Hash access cache (auto)
HardcodeSize	Hardcode depth size	**VoxelCLcache**	Voxel access cache (auto)
CacheVoxels	Voxels block cache	**HardcodeSize**	Hardcode image size
XyzLoopFlat	Flatten inner loop	**UseWi**	Outer loop type
XyzUnroll	Unroll inner loop	**Minmax**	Simplify start/end of ray
EntryIdUnroll	Unroll outer loop		
DepthLocal	Depth map cache		
BranchPos	Condition placement		
ICP		Combined	
Name	Description	Name	Description
PointInterp	Point interpolation	**-**	Same knobs as Depth Fusion
NormalInterp	Normal interpolation	**SdfLocal**	Projection map cache
NablaSumtype	How ∇ is summed		
HessianSumtype	How H is summed		
Branch	Enable branching		
NablaUnroll	∇ sum - unroll factor		
HessianUnroll	H sum - unroll factor		
ShiftRegister	Size of shift registers		

2.5.1 Experimental Setup

We implement the full end-to-end InfiniTAM application on: 1) a Terasic DE1-SoC board with a Cyclone V FPGA and a dual-core ARM Cortex A9 processor, and 2) a Terasic DE5 PCIe board with a Stratix V FPGA, connected to a workstation with an x64 quad-core i7-4790K CPU.

We write all of our kernels using OpenCL, compile them with the Intel FPGA OpenCL SDK v16.1, and integrate them into the *InfiniTAMv2* code base. Each combination of knobs in our kernels produces a unique design. We choose reasonable values for knobs by selecting mostly powers of two and values likely to generate a small design, considering the smallest FPGA. In total, we have compiled more than 480 unique designs across all algorithms for the DE1 FPGA, and more than 2600 for the DE5 FPGA.

For most of our experiments, we fix the input depth map size to *QVGA* (320x240), with a few designs compiled for an input size of 320x180. To test and measure the performance of the kernels, we use the following benchmarks from the standard *TUM* RGB-D SLAM dataset [105]: **fr1/desk**, **fr1/360** and **fr1/room**. We also use a custom dataset (**Cave**) made from data collected with a Google Tango tablet in an underground cave environment. We set the TSDF voxel size to 1 cm, which is comparable to the depth resolution of the sensor used in the benchmarks.

We record the running time of different modules of InfiniTAM. Due to the limited amount of shared memory that can be allocated on the SoC board, our kernels are configured to handle a limited amount of data in the 3D model. We run each benchmark for a specific number of frames (fr1/desk: 595; fr1/room: 700; fr1/360: 440; cave: 400). Below we present the results of this design space exploration, combining FPGA logic utilization and running time.

2.5.2 FPGA SoC Design

Here we describe a fully functional dense SLAM system running on a small Cyclone V FPGA SoC. Consequently, we want to focus on designs that have low resource usage while still maintaining real-time performance. To do this, we perform comprehensive analysis of the design

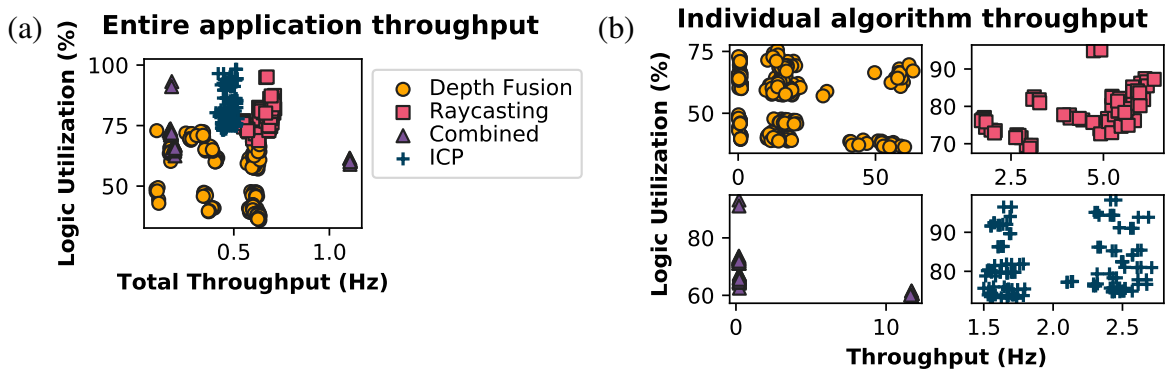


Figure 2.4. Design spaces of InfiniTAM running on the DE1 board with the *Room* benchmark. (a) plots the throughput of the entire application when running the selected algorithm on FPGA; (b) plots the throughput of individual algorithms. The y axis shows the logic utilization of individual algorithms.

spaces for the different dense SLAM components.

Fig. 2.4 illustrates the design spaces on one of our benchmarks (room). Fig. 2.4(a) shows the total frame rate of the application when running different designs of different modules on FPGA, and Fig. 2.4(b) shows the throughput measured individually on each design.

ICP has the worse individual throughput overall and does not contribute to decrease the total running time of the application. The best versions of Depth Fusion and Raycasting both increase the total throughput and the Combined algorithm provides the highest total throughput. This highlights the benefit of combining Depth Fusion and Ray Casting into a single kernel as the individual kernels cannot both fit on the device.

Throughput

Table 2.2 presents the lowest running times for each benchmark, on the dual-core ARM processor with OpenMP (*Baseline*) and with one module accelerated on FPGA. We use *RaycastO3*, which is the Ray Casting algorithm without the interpolation and refine steps. As we implement these optimizations on FPGA, we also transpose them to CPU for a fair comparison. The Combined algorithm is compared to the added time of Depth Fusion and RaycastO3 on ARM.

Table 2.2. Comparison of the lowest running times of different kernels running on the DE1 SoC board. Each row compares the algorithm running on the ARM processor of the DE1 (top, in ms) and on the FPGA (bottom, in ms).

	Cave	Desk	360	Room
Depth Fusion	702.25 18.09 (**38.8**×)	425.75 13.47 (**31.6**×)	573.90 16.80 (**34.2**×)	508.89 15.71 (**32.4**×)
RaycastO3	522.66 35.21 (**14.8**×)	538.92 30.49 (**17.7**×)	455.15 26.53 (**17.2**×)	448.64 28.34 (**15.8**×)
Combined	1224.91 78.29 (**15.6**×)	964.67 89.41 (**10.8**×)	1029.05 85.08 (**12.1**×)	957.53 84.92 (**11.3**×)
ICP	348.35 265.84 (**1.3**×)	537.88 357.57 (**1.5**×)	508.22 330.29 (**1.5**×)	557.56 368.87 (**1.5**×)

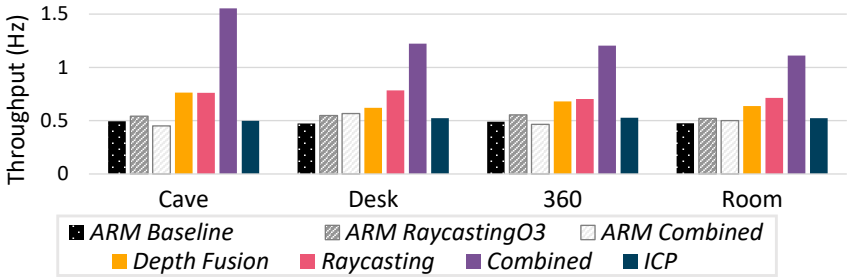


Figure 2.5. Comparison of the average total frame rate between different versions of the application running on the DE1: running on ARM only, or the best versions accelerated on the FPGA. We present the results for different benchmarks.

The Depth Fusion algorithm presents the highest acceleration (up to 38×, or 34× for the TUM benchmarks). ICP is not very well accelerated because the designs implementing shift registers could not fit on this device, but the Combined kernel, replacing two of the three main algorithms, is accelerated more than 10×.

Fig. 2.5 shows the average throughput in frames per second (FPS) for each benchmark. We compare the different baseline versions (without modifications, with RaycastingO3, with the Combined algorithm) and the fastest FPGA versions. We achieve the best throughput with the Combined kernel, but Depth Fusion and Ray Casting are also faster than the baseline. We can achieve up to 1.55 FPS at 320x180, and up to 1.22 FPS at 320x240, which is faster than the CPU version (up to 0.49 and 0.65 FPS respectively).

LASSO Analysis

We perform a statistical analysis to better understand the impact of knobs on each design space. We use the LASSO operator [109] which creates a least square regression model over the knob values that best fits the objective curve (throughput, logic). LASSO can create sparse models by forcing knobs with a small contribution to the model to zero with an alpha parameter (a larger alpha will force more coefficients to zero). We perform a cross-validated search for the alpha that yields a model with the smallest Mean Squared Error (MSE). The result of this analysis is a coefficient for each knob, which represents a relative contribution for this knob to the linear model. A larger coefficient indicates a higher correlation between the knob value and the constructed model. We focus our analysis on the throughput factor and we only present a summary of the results. Our complete data and scripts for a deeper analysis is available in our repository.

For each algorithm, we choose as inputs all the knobs and their 2^{nd} degree polynomial interactions to account for the non-linearity of certain knobs. We vary alpha, compute the model for each alpha, and find the model with the best MSE. Using the knobs without polynomial interactions leads to a higher MSE for most design spaces, which indicates a certain degree of non-linearity in the influence of knobs over the hardware architecture. We present the knobs with the highest coefficients for the models with the lowest MSE in Table 2.3.

The dominant knobs are directly related to the implementation of a loop (*XyzLoop* in Depth Fusion / Combined; *UseWI* in Ray Casting). Using work-items has a great influence on the throughput when compared to using a for loop. The compiler is able to infer a better pipeline with work-items without dependency between iterations and an minimal initiation interval. Other important knobs are generally related to memory caching. Depth Fusion, Raycasting, and Combined all benefit from caching depth or voxel data in the local memory and these mechanisms have a large impact on the running time. The ICP and Combined algorithms are notably different. In the case of ICP, using shift registers to sum large vectors together requires

Table 2.3. LASSO analysis of throughput on the DE1 SoC board for the *Room* benchmark. We take the models with the minimum mean squared error (MSE), and show the 5 knob features with the largest contribution to that model. The knob names are summarized in Table 2.1.

Depth Fusion (Min MSE = 0.0075)		**Raycasting** (Min MSE = 0.0010)	
Knob	Coef	Knob	Coef
XyzLoop^2	0.130	UseWI	0.261
CacheVoxels^2	0.067	VoxelClcache^2	0.152
CacheVoxels	0.031	Minmax	0.075
XyzUnroll^2	0.031	IndexCache,UseWi	0.071
CacheVoxels,XyzLoopFlat	-0.023	HashClcache^2	0.055
ICP (Min MSE = 0.0054)		**Combined** (Min MSE = 10^{-6})	
HessianUnroll	0.336	XyzLoop^2	0.155
NablaUnroll	0.040	XyzLoop	0.051
Branch NablaSumtype	0.023	XyzLoop,VoxelCache	-0.001
NablaSumtype	0.022	XyzFlat	0.001
HessianUnroll,NablaSumtype	0.016	VoxelOutUnroll^2	-0.001

a large amount of resources, and the designs using this optimization cannot fit on the small FPGA. As a result, the remaining designs are very sensitive to the amount of unrolling that is used to paralellize the vector accumulation. This situation leads to a much slower running time (Table 2.2). The Combined kernel is also quite complex and certain memory optimizations do not fit on the Cyclone V FPGA. Consequently, the run time is largely dominated by the loop implementation knob: using work-items speeds up the algorithm, while other knobs impact the running time very little.

2.5.3 PCIe FPGA Design

We compile the algorithms on a DE5 board with a Stratix V FPGA, which is about 7 times larger than the DE1 FPGA (comparing *Logic Elements*, as defined by Intel). We perform a design space exploration to understand how the algorithms scale to a larger device, especially with respect to runtime.

Fig. 2.6 shows the design spaces of individual algorithms and of the entire application running on the *Room* benchmark. The differences with the DE1 design spaces are mostly due to

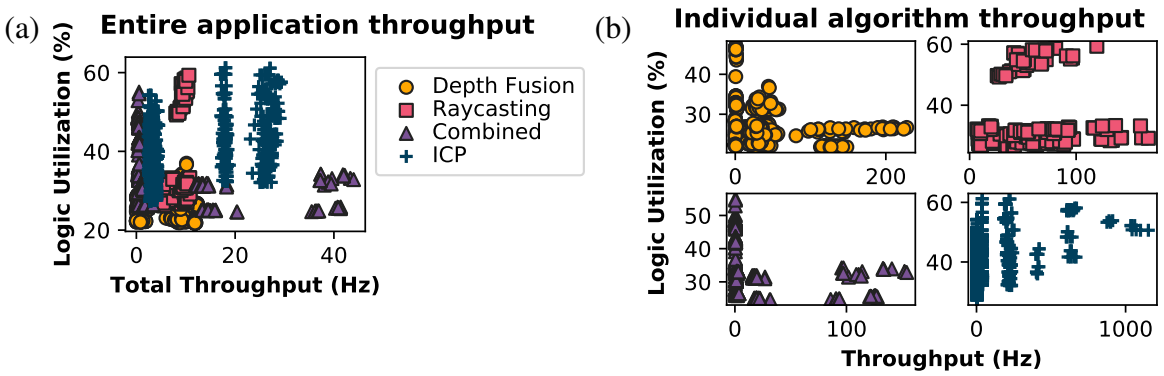


Figure 2.6. Design spaces of InfiniTAM running on the DE5 board with the *Room* benchmark.

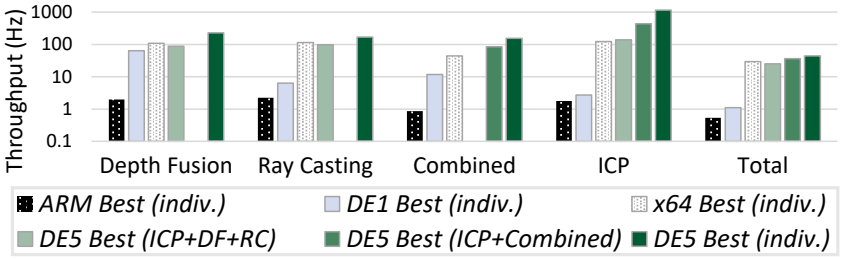


Figure 2.7. Comparison of the best frame rate for individual algorithms and the entire application. We compare the best results on both DE1 and DE5 hardware setups, including the processor and FPGA results. On the DE5, we use bitstreams implementing either multiple algorithms (ICP+DF+RC / ICP+Combined) or only one (indiv.).

the designs that do not fit on the smaller device. The logic utilization is generally low because most of the optimizations were focused on reducing this value to fit on the SoC chip. ICP produces a particularly different design space. There is a clear tradeoff between running time and logic utilization, which means that the most efficient designs have a high utilization and cannot be implemented on a smaller FPGA. The ICP algorithm clearly takes advantage of the increased number of registers on the Stratix V FPGA. Because it is computation-bound, it can be drastically sped up by using a large number of shift registers. Fig. 2.6(a) highlights the most efficient design tradeoffs for the entire application. While the application is not optimized for the DE5 board, we can see that using the Combined designs is generally faster.

We also compare the throughput improvement from the best designs on the DE1 to the designs on the DE5. We extend the design space exploration of individual kernels and compile

Table 2.4. LASSO analysis of throughput on the DE5.

Depth Fusion (Min MSE = 0.010)		**Raycast** (Min MSE = 0.003)	
Knob	Coef	Knob	Coef
$XyzLoop^2$	0.047	$HashCLCache^2$	0.152
XyzLoop,DepthLocal	-0.039	UseWI	0.116
ComputeUnits,DepthLocal	-0.039	Minmax	0.088
EntryidLoop,DepthLocal	-0.034	$IndexCache^2$	0.085
CacheVoxels	0.029	$Minmax^2$	0.055
ICP (Min MSE = 0.007)		**Combined** (Min MSE = 0.0016)	
HessianSumtype	0.056	$XyzLoop^2$	0.061
NablaSumtype	0.054	SdfLocal,XyzLoop	-0.024
NablaSumtype,ShiftRegister	0.047	SdfLocal,XyzUnroll	-0.012
HessianUnroll	0.038	SdfLocal,XyzFlat	0.012
HessianUnroll,HessianSumtype	0.033	XyzUnroll	-0.010

two other types of bitstreams: one containing Depth Fusion, Ray Casting and ICP (DF+RC+ICP), and one with the Combined kernel and ICP (Combined+ICP). A full design space exploration of multiple kernels would lead to millions of unique combinations, so we only compile a few combinations of Pareto-optimal designs. We report the best performance of each algorithm for the different types of bitstreams (DF+RC+ICP, Combined+ICP), and individual algorithms in Fig. 2.7. In this figure, we compare the best throughput results between the DE1 FPGA, DE1 processor, DE5 FPGA, and DE5 processor, for individual algorithms and the entire application on the *Room* benchmark. Depth Fusion is improved by 3.6×, Ray Casting 4.8×, Combined 13.1×, and ICP is accelerated by 425.2×. The entire application runs at up to 44 FPS.

LASSO Analysis

We perform a LASSO analysis on these design spaces to better understand the importance of knobs on larger and more complex architectures. Table 2.4 presents the summary of our LASSO analysis on the *Room* benchmark. In general, the throughput is harder to model on these larger spaces. There are more knob values and interactions, and as a result, the MSE is slightly higher for all algorithms. The general principles defined from the DE1 LASSO analysis

still apply, as the type of loop implementation still has a great influence on the throughput, and memory caching is another useful optimization. The Combined algorithm has a large data dependency which is removed when using work-items, which tends to change the accuracy result. ICP is much less memory bound, and this fact is reflected in the knob coefficients, which tend to give weights to the knobs affecting the summation of data.

2.5.4 Real-Time Experiments

To test our implementation in a real environment using a complete system, we use a Google Tango tablet as both a depth camera and a screen to display real-time feedback. The Google Tango implements a light network client that only sends depth data and receives an image for display. The DE1 board receives, processes, and sends the data back through the wired/wireless network. We run the fastest Combined kernel from our design space exploration on the FPGA. We scan an office desk for about 1 minute (100 frames) and obtain the result shown in Fig. 2.1. In average, each frame took 505 ms to process (almost 2 FPS).

2.6 Conclusion

We have described implementations of dense SLAM on an FPGA, outlining the potential hardware optimizations for the sub-algorithms (Tracking, Depth Fusion, and Ray Casting). We have parameterized our designs to create a vast set of FPGA architectures, and analyzed the resulting design spaces to adapt them to two different FPGA platforms. We have built functional dense SLAM systems on two FPGA platforms. Our complete end-to-end system on the FPGA SoC achieves up to 2 FPS, and 44 FPS on a higher performance PCI-e FPGA. We expect that these numbers would be largely increased by using newer SoC boards currently on the market.

Design space exploration has provided insightful information about hardware tradeoffs for dense SLAM. Our analysis showed that the representation of loops in OpenCL for FPGA has a profound impact on the run time. Additionally, it benefits greatly from memory cache optimizations.

Our architectures for InfiniTAM are made open-source, which includes our full system implementations, OpenCL code, and design space exploration results. These results can be useful to gain valuable insight into the implementation of complex SLAM applications on different FPGA hardware, but also to understand the mapping between optimization knobs and the final design spaces. Ultimately, this knowledge can help develop automated tools for design space exploration to avoid the manual optimization process for designers.

Future Work

With recent improvements in deep learning algorithms, dense SLAM can now be formulated as a learning problem and can provide results on par with traditional SLAM techniques [106, 128]. Additionally, there has been much progress in the implementation of deep neural networks on FPGA [123, 116, 113]. Future work can explore the implementation of neural network-based SLAM, and analyze the effects of hardware knobs as well as software knobs (network depth, etc.) on the SLAM results.

Acknowledgments

Section 2.3, in part, is a reprint of the material as it appears in the International Conference on Field-Programmable Technology (FPT) 2014. "Real-time 3d reconstruction for fpgas: A case study for evaluating the performance, area, and programmability trade-offs of the altera opencl sdk." Gautier, Quentin; Shearer, Alexandria; Matai, Janarbek; Richmond, Dustin; Meng, Pingfan; and Kastner, Ryan. The dissertation author was the primary investigator and author of this paper.

Chapter 2, in part, is a reprint of the material as it appears in the International Conference on Application-specific Systems, Architectures and Processors (ASAP) 2019. "FPGA Architectures for Real-Time Dense SLAM." Gautier, Quentin; Althoff, Alric; and Kastner, Ryan. The dissertation author was the primary investigator and author of this paper.

Chapter 3

Spector: An OpenCL FPGA Benchmark Suite

3.1 Introduction

FPGA design was traditionally relegated to only experienced hardware designers, and required specifying the application using low-level hardware design languages. This provides opportunities to create highly specialized custom architectures; yet it is time consuming as every minute detail must be specified on a cycle-by-cycle basis. Recently, FPGA vendors have released high-level synthesis tools centered around the OpenCL programming model. The tools directly synthesize OpenCL *kernels* to programmable logic creating a custom hardware accelerator. They raise the level of abstraction of the programming model and increase the designer's productivity. Furthermore, the tools manage the transfer of data between the FPGA and the CPU host. This opens the door for more programmers to easily utilize FPGAs.

OpenCL is a open standard that provides a framework for programming heterogenous systems. The language extends C with features that specify different levels of parallelism and define a memory hierarchy. There exists OpenCL implementations for a variety of multicore CPUs, DSPs, and GPUs. More recently, commercial tools like Xilinx SDAccel [5] and the Altera OpenCL SDK [3] add FPGAs into the mix of supported OpenCL devices. This greatly simplifies the integration of FPGAs into heterogeneous systems, and provides a FPGA design entry point for a larger audience of programmers.

The OpenCL FPGA design process starts with implementing the application using OpenCL semantics. The designer then typically employs some combination of well-known optimizations (e.g. varying the number of work-items, using SIMD vectors, loop unrolling, etc.) and settles on a small set of designs that are considered optimal according to some metric of performance (resource utilization, power, etc.). Most designers will need multiple attempts with several optimization options to understand the design space. Unfortunately, a major drawback of these OpenCL FPGA tools is that the compilation time is long; it can take hours or even days. This severely limits the ability to perform a large scale design space exploration, and requires techniques to efficiently guide the designer to a good solution.

In many applications, it is difficult to predict the performance and area results, especially when optimization parameters interact with each other in unforeseen manners. As an example, increasing the number of compute units duplicates the OpenCL kernel, which should improve performance at the expense of FPGA resources. However, this is not always true as memory contention may limit the application's performance. Finding when this occurs requires a better understanding of the memory access patterns and how other optimizations alter it. Many other optimizations are also intertwined in non-intuitive ways as we describe throughout the paper.

We propose an OpenCL FPGA benchmark suite. Each benchmark is tunable by changing a set of *knobs* that modify the resulting FPGA design. We compiled over 8000 designs across 9 unique benchmarks using the Altera OpenCL SDK. All of our results are openly available and easily accessible [4]. This provides large sets of designs to enable research on system-level synthesis for FPGAs. These results can be used to evaluate methods for improving the process of design space exploration. We provide our own analysis of the data to show that the exploration of such designs is not always an easy task.

The major contributions are:

- Designing and releasing an OpenCL FPGA benchmark suite
- Creating an optimization space for each benchmark and describing the parameters that

define it.

- Performing a comprehensive set of end-to-end synthesis experiments, the result of which is over twenty thousands hours of compilation time.
- Providing a statistical analysis on the results to give insights on OpenCL FPGA design space exploration.

The remainder of this chapter is organized as follows. In Section 3.2, we motivate the need for this research, and discuss related work. In Section 3.3 we detail our benchmark design process and talk about how we obtained the results. In Section 3.4 we describe the benchmarks, detail the tunable knobs and their effect on the architectures, and present their design spaces. We give a statistical analysis of some of the results in Section 3.5 and conclude in Section 3.6.

3.2 Motivation

There are substantial number of application case studies for parallel computing, heterogeneous computing, and hardware design. One can start with reference designs from hardware vendors such as Intel, NVIDIA, Altera, and Xilinx. Unfortunately, these are often scattered across different websites, use different target compute platforms (CPU, GPU, FPGA), and they lack a common lingua franca in terms of optimization parameters. This makes them difficult to provide a fair comparison across the different applications. This is the general motivation for benchmark suites – they provide highly available, well documented, representative set of applications that can assess the effectiveness of different design strategies and optimizations.

Several open-source benchmarks for parallel applications currently exist. Many of these, e.g., the HPEC challenge benchmark suite [48] or Rodinia [24], focus on GPUs and multicore CPUs. FPGAs have a different compute model. Thus, while some of the applications in these benchmarks suites are applicable to studying the OpenCL to FPGA design flow, they require modifications to be useful. Several of our benchmarks are found in these existing benchmark suites.

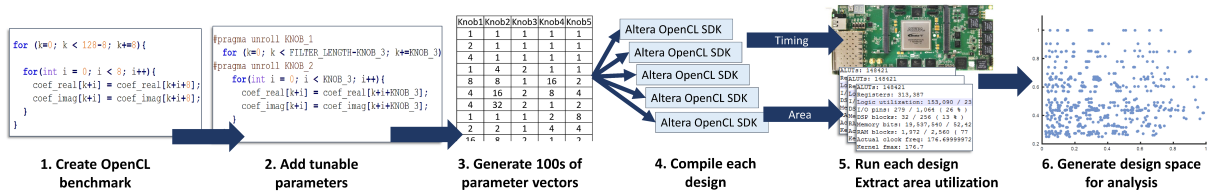


Figure 3.1. Our workflow to generate each benchmark and the design space results.

There are a number of FPGA specific benchmarks suites. These generally target different parts of the design flow. For example, the applications in ERCBench [21] are written in Verilog and useful for studying RTL optimizations or as a comparison point for hardware/software partitioning. Titan [83] uses a customized workflow to create benchmarks to study FPGA architecture and CAD tools. The OpenCL dwarfs [38, 63] contain several OpenCL programs that have been optimized for FPGA. Unfortunately, they usually have a fixed architecture with little to no optimization parameters.

We seek to extend these benchmark suites by leveraging the existing OpenCL benchmarks and reference programs, and outfitting them with multiple optimization parameters. Each of these designs can be compiled with a commercial program or open-source tool to generate thousands of unique configurations. We make open-source our results that we obtained from the Altera software, and encourage the community to compile our benchmarks with different tools. One of our motivating factors in creating this benchmark suite was the lack of a common set of designs and optimization parameters for comparing different design space exploration (DSE) techniques. Machine learning techniques for DSE in particular can benefit from a large set of designs, e.g., [130] and [74] use machine learning approaches to explore design spaces and predict the set of Pareto designs without having to compile the entire space. These techniques could directly leverage our results to verify and perhaps improve their models. And in general, we believe that open repository of OpenCL FPGA designs will benefit this and other areas of research.

Table 3.1. Number of successfully compiled designs.

BFS	507	Histogram	894	Normal estimation	696
DCT	211	Matrix Multiply	1180	Sobel filter	1381
FIR filter	1173	Merge sort	1532	SPMV	740

3.3 Methodology

We designed nine benchmarks that cover a wide range of applications. We selected benchmarks that are recurrent in FPGA accelerated applications (FIR filter, matrix multiply, etc.), but we also included code with more specific purpose to cover potential real-world programs (like histogram calculation and 3D normal estimation). These benchmarks come from various places: some were written directly without example source code, others come from GPU examples, and some come from FPGA optimized examples. In all cases, we started from programs that contained little to no optimization parameters, thus requiring us to define the optimization space.

For each benchmark, we proceeded as illustrated in Figure 3.1. First we created or obtained code that was partially or fully optimized for FPGA. It is important to note that we were not trying to reach a single "most optimal" design, but instead defining an optimization space that covers a wide range of optimizations. We studied which types of optimization would be relevant for each benchmark. Then we added several optimization *knobs*, which are values that we can tune at compile-time. These knobs can enable or disable code, or affect an optimization parameter (e.g., unrolling factor). We compiled several sample designs to ensure that the knobs we had chosen would have *some* impact on the timing and area. Each benchmark has a set of scripts to generate hundreds of unique designs, with all the possible combinations of knob values. In most cases we restricted the values of the knobs to a subset of the options (e.g., powers of two). We also removed values that were likely to use more resources than available, and filtered out further using pre-*place-and-route* estimations. All the benchmarks were written using standard OpenCL code with a C++ host program that can run and measure the time of execution. The OpenCL code works with the Altera SDK for FPGA, and can also be executed on GPU

and CPU. Although we have not tested the programs with other commercial or open-source OpenCL-to-FPGA pipelines, we expect that little to no modifications are required to ensure compatibility.

Each design was then individually compiled using the Altera OpenCL SDK v14.1, with a compile time typically requiring 1 to 4 hours on a modern server, and occasionally taking more than 5 hours per design. In total we successfully compiled more than 8300 designs (see Table 3.1), plus many more that went through almost the entire compilation process but failed due to device resource limits. We executed the successful designs on a Terasic DE5 board with a Stratix V FPGA to measure the running time for a fixed input. Some applications can behave differently given a different set of input data however, and the optimizations to use in these cases might vary. This is the case for algorithms like graph traversal or sparse matrix multiplication, where the sparsity of the input can have a significant impact on the design space. In both of these benchmarks we ran the program with two sets of inputs. We ran BFS with both a densely connected graph and one with sparse edges. Sparse matrix-vector multiplication was run with one matrix containing 0.5% of non-zero values and one only 50% sparse. We extracted the running time and area data (such as logic, block RAM, DSPs, etc.) for each run. The set of design spaces that we present in Figure 3.2 shows the logic utilization against the running time, as logic is usually the most important resource and often the limiting factor. These results are generated from the scripts *plot_design_space.m* and *plot_all_DS.m* available in our repository.

3.4 Benchmarks Description

Here we describe the benchmarks and the knobs that we have chosen, so that the reader can interpret the design space results based on the design choices. First we explain some of the most common optimization types in OpenCL designs. Then we explain each benchmark in more details, followed by an overview of the shape of the design space.

Work-items: These are parallel threads with a common context on GPU. On FPGA,

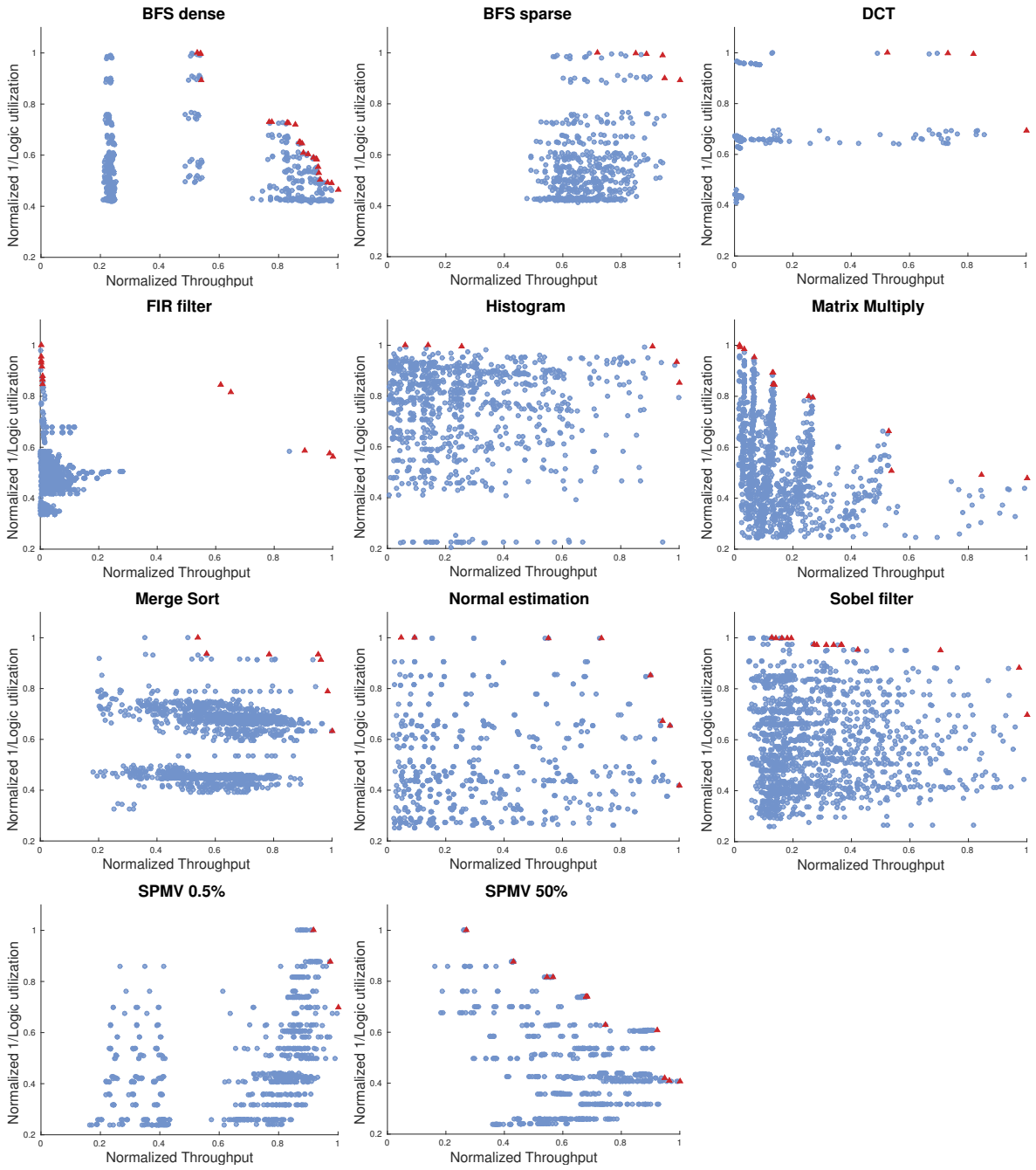


Figure 3.2. Normalized design space for each benchmark. We plot the inverse logic utilization against the throughput such that higher values are better. The Pareto designs are shown in red triangles.

they can be interpreted as multiple iterations of an outer loop that is pipelined by the compiler.

Using only one can give more flexibility to the programmer to control unrolling and pipelining,

while using multiple can enable optimizations such as SIMD. **Work-groups**: This defines how many groups of work-items to use, each group using a different context (no shared memory). This is useful to enable the compute units optimization. **Compute units**: How many duplicates of the kernel are created on the FPGA chip. Compute units can run work-groups in parallel, however they all access the external memory and thus might be limited by the bandwidth. **SIMD**: Work-items can be processed simultaneously by increasing local area usage. It is only possible when there is no branching. **Unrolling**: By explicitly unrolling a loop, we can process multiple elements simultaneously by using more area, usually storing data in more local registers.

3.4.1 Breadth-First Search (BFS)

This code is based on the BFS FPGA benchmark from the OpenDwarfs project [38, 63], and originally based on the BFS benchmark from the Rodinia Benchmark Suite [24]. It is an iterative algorithm that simply traverses a graph starting at a specified node by performing a breadth-first traversal, and returns a depth value for each node. The algorithm iterates over two OpenCL kernels until all the reachable nodes have been visited. Each kernel launches work-items for each node in the graph and uses binary masks to enable computation. There are 6 knobs with varying values in these kernels.

- **Unroll factor** in kernel 1: Unrolls a loop to process multiple edges simultaneously. We enable additional code for edge cases only if the unroll factor is greater than 1.
- **Compute units** in kernel 1 and 2, **SIMD** in kernel 2.
- **Enable branch** in kernel 1: Describes how to check if a node was visited and how to update graph mask values. Either enables the code from OpenDwarfs with bitwise operators to avoid branching, or enables the code from Rodinia with regular *if* statement.
- **Mask type**: Number of bits used to encode the values of graph masks.

Design space

The design space for the dense input is clearly divided along the timing axis. The left cluster represents the designs where the branching code is disabled. The more optimized branching code gets better performance as we increase the unrolling factor. The discontinuity between the middle and right clusters is caused by a jump in the knob values. The impact of the unrolling factor is however limited by the number of edges to process per node. This limitation is reflected in the sparse input design space that is more uniform.

3.4.2 Discrete Cosine Transform (DCT)

This algorithm is based on the NVIDIA CUDA implementation of a 2D 8x8 DCT [90]. The program divides the input signal into 8x8 blocks loaded into shared memory, then processed by calculating DCT for rows then columns with precalculated coefficients. Some knobs enable multiple blocks to be loaded in shared memory, and multiple rows and columns to be processed simultaneously. Each work-group processes one 8x8 block, and within the group each work-item processes one row and column. This can be altered by 9 tunable knobs:

- **SIMD** and **Compute units**.
- **Block size**: Number of rows/columns to process per work-item.
- **Block dim. X** and **Block dim. Y**: Number of blocks per work-group in X or Y direction.
- **Manual SIMD type**: Using OpenCL vector types, each work-item processes either multiple consecutive rows/columns, or processes multiple rows/columns from different blocks.
- **Manual SIMD**: Number of rows/columns for one work-item to process using SIMD vector types.
- **Unroll factor**: Unroll factor for the loops launching 8-point DCT on rows and columns.

Table 3.2. FIR filter Pareto optimal designs.

Coef. shift	8	8	1	8	1	1	8	1	8	1	8	1
Num. parallel	1	2	2	1	1	4	2	2	2	2	1	1
Unroll inner	32	32	32	32	32	1	2	2	1	1	2	2
Unroll outer	2	1	1	1	1	1	1	1	1	1	1	1
Work-items = Work-groups = SIMD = Compute units = 1												
Time (ms)	0.70	0.71	0.78	1.08	1.14	80.73	85.35	94.93	151.9	159.9	190.1	192.2
Logic	52%	50%	50%	36%	34%	34%	34%	33%	32%	31%	31%	30%

- **DCT unroll**: Either use the loop version of 8-point DCT, or the manually unrolled 8-point DCT.

Design space

The designs are clearly divided into clusters along the logic utilization axis. These clusters can be mostly explained by the *block size*, *manual SIMD* and *compute units* knobs that have a similar impact on logic. The combination of these knobs increases by steps, creating the clustering of the design space.

3.4.3 Finite Impulse Response (FIR) Filter

This benchmark is based on the Altera OpenCL design example of a Time-Domain FIR Filter, itself based on the HPEC Challenge Benchmark suite [48]. This code implements a complex single-precision floating point filter, where multiple filters are applied to a stream of input data using a sliding window. After a block of input data has been processed, it loads the next filter's coefficients while still shifting the sliding window to avoid too much branching complexity. The kernel is originally a single work-item task, but it has been extended to use multiple work-items. There are 8 tunable knobs:

- **Coefficient shift**: Number of filter coefficients to load at each loading iteration.
- **Num. parallel**: Number of FIR computations to perform in a single iteration. This extends the size of the sliding window.

- **Unroll inner**: Unroll factor for the FIR computation loop (for each coefficient).
- **Unroll outer**: Unroll factor for the main loop (for each input value).
- **Work-items** Number of work-items. This divides the input data into multiple blocks, each work-item works on one block.
- **Work-groups**: Number of work groups, same consequences as work-items, but also enable compute units.
- **SIMD** and **Compute units**.

Design space

The FIR filter benchmark has this particularity to present a small group of outlier results that turn out to be the most efficient designs. Unsurprisingly, the values of the knobs correspond to the original code from Altera that is thoroughly optimized for FPGA. It's a single work-item sliding window with a fully unrolled filter computation, loading 8 complex numbers when loading a new filter (8x2x32 bits = 512 bits, the external memory width). The difference with the original is the unrolling or sliding window size that are bigger, allowing two elements per iteration to be processed. This is possible because we use a smaller filter size than the original. As we follow the Pareto front toward less logic utilization (Table 3.2), we simply unroll less the computation, and decrease the sliding window size. From this design space we can also learn that, even though not Pareto optimal, the next most efficient designs come from using pipelining with multiple work-items. It is less efficient due to accesses to non-contiguous portions of the external memory.

3.4.4 Histogram

This code calculates the distribution histogram of unsigned 8-bits values by calculating the number of each of the 256 unique values. One OpenCL kernel counts the input values into a local histogram. If multiple work-items are used, we divide the input data and calculate multiple

histograms that can be combined either in shared memory, or through global memory with a second kernel. The second kernel uses a single work-item to sum them locally and output the result. The first kernel can also process multiple values at the same time by using several local histograms. There are 7 tunable knobs:

- **Num. histograms**: This is the number of local histograms in the first kernel to compute simultaneously.
- **Histogram size**: Switches between local histogram storage in registers or in block RAM.
- **Work-items**: This will create intermediate results that need to be accumulated.
- **Work-groups**: If there are multiple work-groups, the intermediate results have to be accumulated in a second kernel (cannot use shared memory).
- **Compute units**.
- **Accum. shared memory**: Choose to accumulate the intermediate results in shared memory if possible (only one work-group), or in a second kernel.
- **Unroll factor** Unrolls the main loop over the input values.

Design space

This is one of the few design spaces that appear relatively uniform. All the knobs seem to have a similar impact on the variations between designs, although a few design make the exception by using more logic. These designs set the *histogram size* such that the compiler will prefer to use registers instead of block RAMs, as they are using only one local histogram. But as opposed to some other uniform design spaces, the parameters cannot be linearly modeled, as presented in Section 3.5.

3.4.5 Matrix Multiplication

This code is based on an Altera OpenCL example. It implements a simple matrix multiply $C = A * B$ with squared floating-point matrices. The matrix C is divided into blocks, each computed individually. The implementation includes several knobs to change the size of the blocks and to process multiple blocks at once. Each work-group takes care of one block of C. Each work-item takes care of one element in a block, including loading elements to local storage, multiplying one row of block A by one column of block B, and copying back to global storage. There are knobs that enable multiple block processing for work-groups and work-items, either by adding an inner loop, or by using OpenCL vector types for SIMD computation. There are 9 tunable knobs in this code:

- **Block dimension**: Width of blocks.
- **Sub-dimension X** and **Sub-dimension Y**: How many blocks of C in X or Y direction to process in one work-group. This adds a for loop so that each work-item processes this number of blocks.
- **Manual SIMD X** and **Manual SIMD Y**: How many blocks of C in X or Y direction to process in one work-group. This performs the inner matrix multiply with OpenCL vector types.
- **SIMD** and **Compute units**
- **Enable unroll**: Enables or disables unrolling loop on load and store operations.
- **Unroll factor**: Unroll factor for multiple loops.

Design space

The matrix multiplication Pareto-optimal designs are a good example of an almost linear relationship between area and timing (in this optimization space). By looking at the knob values along the Pareto front, we can determine that it's mostly a combination of *block dimension*,

manual SIMD X, *SIMD*, and *unroll factor* that can vary the results and create the trade-off between speed and area. The other knobs still have an impact on the design space, but tend to have a single optimal value for both area and timing. Typically, *manual SIMD Y* is not enabled in the optimal designs, as the data are organized along the X direction.

3.4.6 Merge Sort

This program applies the Merge Sort algorithm using loops, merging in local memory first, then in global memory:

```
1:  for each chunk of size localsortsize do
2:      copy the entire chunk into local memory
3:      for localchunksize = 2 to localsortsize do
4:          for each local chunk of the chunk do
5:              merge two halves of local chunk
6:          end for
7:          swap input/output buffers
8:      end for
9:  end for
10: for chunksize from localsortsize to inputsize do
11:     for each chunk of the input do
12:         merge two halves of the chunk
13:     end for
14:     swap input/output buffers
15: end for
```

- **Work-items**: Each work-item processes a different chunk in lines 4 and 11.
- **Local sort size**: Varies $localsortsize$.
- **Local use pointer**: Use pointers to swap buffers in local memory. This can force the use of block RAMs instead of registers.

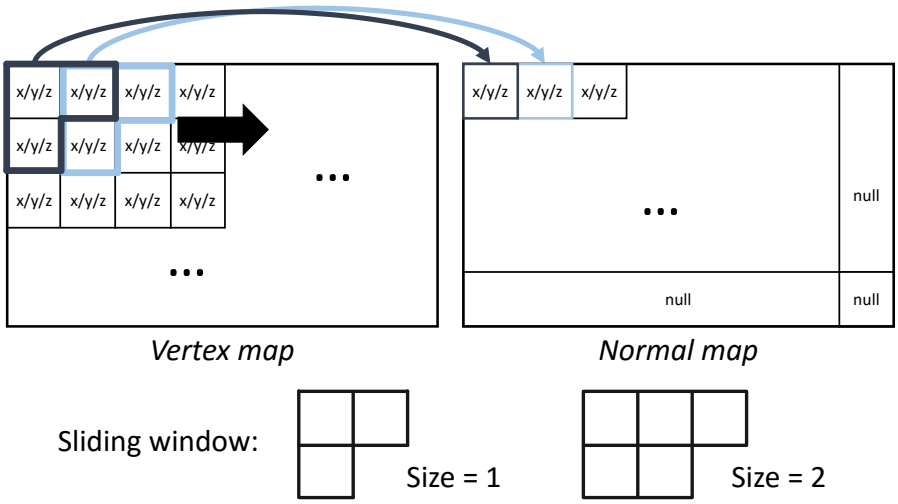


Figure 3.3. Estimating 3D normals from 3D vertices organized on a 2D map. The top of the figure shows how the sliding window works in the algorithm. The bottom illustrates how the sliding window can be tuned.

- **Specialized code**: Enable a specialized code to merge chunks of size 2.
- **Work-groups**: Each work-group runs the algorithm on one portion of the input data. A second iteration of the kernel is launched to merge the final output.
- **Compute units**
- **Unroll**: Unroll factor for the loops copying data from/to local memory.

Design space

This design space is mainly divided into 2 clusters, due to the *compute units* knob that can take the value 1 or 2. In this case, using multiple compute units has a large impact on the resource utilization, while the other knobs have a much smaller impact on resources, and are responsible for smaller variations within each cluster. Interestingly, the fastest designs use only one compute unit, but make use of the pipeline optimization from work-items.

3.4.7 3D Normal Estimation

This code is inspired by an algorithm in the KinectFusion code from the PCL library [97]. It estimates the 3D normals for an organized 3D point cloud that comes from a depth map (vertex map). We can quickly estimate the normals for all the points by using the right and bottom neighbors on the 2D map, then calculate the cross-product of the difference between each neighbor and the current point, and normalize. If any of the vertices is null, the normal is null. One kernel does the entire computation using a small sliding window where the right neighbor is shifted at each iteration to be reused in the next iteration. As illustrated in Figure 3.3, a parameter can vary the window size so that multiple inputs are processed in one iteration. If multiple work-items are used, the input data are cut into blocks of whole rows. There are 6 tunable knobs:

- **Work-items**, **Work-groups** and **Compute units**.
- **Unroll factor 1**: Unroll factor for the outer loop that iterates over all the input data.
- **Unroll factor 2**: Unroll factor for the loop that iterates over the elements within a sliding window.
- **Window size**: Size of the sliding window. *ie.* number of consecutive elements to process in one iteration.

Design space

Normal estimation is another example of a fairly uniform design space. It is easier to create a model of the knobs (see Section 3.5), and it is a good example of a optimization space where most parameters have an impact of similar importance on both the timing and the area utilization.

3.4.8 Sobel Filter

This code applies a Sobel filter on an input RGB image, based on the Altera OpenCL example.

```
1:  for each block on the input image do
2:    load pixel values from block in shared memory
3:    for each pixel in local storage do
4:      load the 8 pixels around from shared memory to registers
5:      convert pixels to grayscale
6:      apply the 3x3 filter in X and Y
7:      combine the X and Y results and apply threshold
8:      save result in global storage
9:    end for
10: end for
```

Work-group take care of blocks (line 1) and work-item take care of pixels within the block (line 3). The knobs can enable a sliding window within the blocks, SIMD computation, or make a work-item perform multiple computations. With the SIMD parameter, each work-item loads more pixels to registers to apply multiple filters by using OpenCL vector types. The sliding window parameter creates an inner loop (after line 3) where each work-item processes one pixel (or multiple with SIMD), then shifts the registers to load one new row or column of data from the local storage. There are 8 knobs in this code:

- **Block dimension X** and **Block dimension Y**: Size of each block in X or Y.
- **Sub-dimension X** and **Sub-dimension Y**: Local sliding window size, moving in X or Y direction.
- **Manual SIMD X** and **Manual SIMD Y**: Number of elements to process as SIMD in X or Y direction.
- **SIMD** and **Compute units**.

Design space

The Sobel filter is another example of a mostly uniform design space where all the knobs seem to have a similar impact on the output variables. A more detailed look at the knob values actually shows that along the timing axis, the *manual SIMD X* knob is one of the most important factors, and the most important on the Pareto front. This is a case where manually designing SIMD computation is better than using automatic SIMD, and this becomes apparent from the analysis of an entire optimization space.

3.4.9 Sparse Matrix-Vector Multiplication (SPMV)

This code is also based on an OpenDwarfs benchmark. It calculates $Ax + y$ where the matrix A is sparse in CSR format and the vectors x and y are dense. Each work-item processes one row of A to multiply the non-zero elements by elements of x. Two knobs control the number of elements processed simultaneously by one work-item: one unrolls a loop, and the other enables the use of OpenCL SIMD vector types to store and multiply the data. There are 4 tunable knobs:

- **Block dimension**: Number of work-items per work-group.
- **Compute units**.
- **Unroll factor**: This creates an inner loop over some number of elements and unrolls it so that elements can be processed simultaneously.
- **Manual SIMD width**: This is the size of the OpenCL vector type to use when processing elements. Elements are loaded and multiplied in parallel using this type.

Design space

This benchmark is dependent on the type of input and behaves differently for more or less sparse matrices. This is reflected in the design spaces, where the most efficient designs for sparse matrices, and particularly the Pareto optimal designs, tend to have smaller values for

Unroll factor and *Manual SIMD width*. When processing a denser matrix, the best designs tend to have a higher value for these knobs, as it allows simultaneous processing of elements in rows. For sparse matrices, the pipelining provided by *block dimension* is usually preferred to the SIMD and unroll optimizations.

3.5 Design Space Analysis

To demonstrate one potential use of our data, we perform an example analysis to determine the viability of multiple sparse linear regression to model design space performance and area. In mathematical form we compute model coefficients β in

$$f(x) = \beta_0 + \sum_{i=1}^{n} x_i \beta_i + \sum_{i=n+1}^{m} \sum_{j=i+1}^{m} x_i x_j \beta_{i \cdot m + j} \tag{3.1}$$

where x is a vector of design space knob values, n is the number of design space knobs, and the number of entries in β is $m = n(n+1)/2$. In the following sections, we use the term "parameters" to refer to values of β and "realization" to refer to a single design (a single point in the design space plots of previous sections).

The purpose of this analysis is *not* to suggest that linear regression is a good idea when seeking to model a design space in general. Rather we observe that there are many cases where simple linear models are effective, *and equally many* where they are misleading and/or downright ridiculous. The lesson here is that DSE research involving parametric models should not overstate their generality, particularly where performance is concerned.

3.5.1 The Least Absolute Shrinkage and Selection Operator (LASSO)

The LASSO is a well-known statistical operator [109] useful for variable selection and sparse modeling. While we present its mathematical form in equation (3.2), LASSO is, in essence, ordinary least squares regression with a penalty forcing small variables toward zero. The operator parameter λ determines "small", and it is often—as it is in our case—selected

via cross-validation. A small value at β_k indicates that variation of the kth parameter does not produce significant variation in the output. We prefer LASSO for this analysis because it tends to produce simpler and more interpretable models. The LASSO in mathematical form is

$$\begin{aligned} &\min_{\beta} \|y - X\beta\|_2^2 + \lambda \|x\|_1 \\ &= \min_{\beta} \sum_{i=1}^{n} (y_i - \sum_{j=1}^{m} X_{ij}\beta_j)^2 + \lambda \sum_{i=1}^{m} |\beta_i| \end{aligned} \tag{3.2}$$

where X_{ij} is the entry of the matrix X at row i and column j. In the remainder of this section β refers to the vector minimizing the LASSO for a λ minimizing the model mean squared error.

While LASSO explicitly determines coefficients for a linear model, it is also useful for variable selection in nonlinear systems [64]. In this situation we do not read very deeply into β, but rather use it to detect when simple linear, perhaps even obvious, relationships exist between parameters and the realized design space. To summarize the LASSO results we compute the coefficient of determination—also known as the r^2 value—independently for throughput and area, denoted r_t^2 and r_ℓ^2 respectively. r^2 is a commonly used goodness-of-fit measure indicating the amount of variance in the data explained by the model. Alongside $r_{t,\ell}^2$ we also compute the Gini coefficient of β, $G_{t,\ell}(\beta)$, as a measure of model complexity. Note that if r^2 is small then $G(\beta)$ is a nearly worthless quantity. We do, however, include values for all design spaces in Table 3.3 for completeness.

3.5.2 Gini Coefficient

The Gini coefficient [46] G is a statistic frequently used to measure economic inequality. G takes values in the range $[0,1]$. If $G(v) = 1 - \varepsilon$ for a particular vector v and small ε, then there are a few elements of the set that are very large relative to others. If $G(v) = \varepsilon$ then all vector elements have values that are close to each other.

Equation (3.3) describing G is calculated using equation (3) with bias correction from

Table 3.3. LASSO r^2 and $G(\beta)$ values for logic (ℓ) and timing (t) across benchmarks for complete and near-Pareto spaces

	Complete Space				**Within 0.1 of Pareto**			
Benchmark	r^2_ℓ	$G_\ell(\beta)$	r^2_t	$G_t(\beta)$	r^2_ℓ	$G_\ell(\beta)$	r^2_t	$G_t(\beta)$
BFS (Dense)	0.89	0.91	0.97	0.97	0.92	0.85	0.83	0.97
BFS (Sparse)	0.89	0.91	0.85	0.84	0.98	0.68	0.92	0.80
DCT	0.98	0.83	0.91	0.86	0.58	0.86	0.95	0.86
FIR	0.61	0.92	0.37	0.94	0.79	0.95	0.99	0.96
Histogram	0.73	0.90	0.04	0.80	-0.05	1.00	0.15	0.94
Matrix multiply	0.83	0.76	0.70	0.70	0.91	0.82	0.94	0.71
Normal estimation	0.90	0.81	0.91	0.57	0.98	0.72	0.98	0.69
Sobel	0.78	0.77	0.88	0.67	0.98	0.83	0.94	0.78
SPMV (Sparse)	0.88	0.93	0.29	0.71	0.90	0.89	0.24	0.93
SPMV (Dense)	0.88	0.93	0.24	0.82	0.90	0.94	0.68	0.80
Mergesort	0.91	0.89	0.43	0.68	0.95	0.92	0.74	0.81

Note: $G(\beta)$ values for which the associated $r^2 < 0.7$ are greyed out to indicate that they should be disregarded

[30]

$$G(x) = \frac{n}{n-1} \cdot \frac{\sum_{i=1}^{n}(2i-n-1)x_i}{n\sum_{i=1}^{n} x_i} \tag{3.3}$$

where x is sorted beforehand. $G(\beta)$ indicates whether variation in the realized design space can be accounted for by a few parameters.

3.5.3 Example observations

To give the reader some idea of the sort of descriptive statistical information that can be gained from these data, we will consider only the the r^2 and Gini values from Table 3.3 for the *Histogram* and *Normal estimation* design spaces. A purely visual inspection of the design space graph, (see Figure 3.2) might suggest that there is a "grid-like" quality to the relationship between knobs and the realized design space. We demonstrate that this is not necessarily the case.

Histogram

Considering the complete Histogram design space, $r_\ell^2 = 0.73$ where r_ℓ^2 indicates goodness-of-fit on the logic axis. This means that there is a $1 - r^2$ fraction of the total variance unaccounted for by the model, so 0.73 indicates a reasonable—but not excellent—fit for the LASSO model. $G_\ell(\beta) = 0.9$ tells us that the LASSO model, with learned parameter vector β, has a few dominant parameters, while the remainder have negligible influence over logic utilization. On the other hand, $r_t^2 = 0.04$ means that the performance of the design is not well represented as a linear model of the input parameters. For this reason $G_t(\beta)$ should not be taken seriously as an indicator of parameter dominance. Examining these values for the subset of designs within 0.1 of the Pareto front tells a different story. r^2 for both logic and performance are extremely low. While r_t^2 increased—meaning the design space becomes more amenable to linear modeling nearer to the Pareto front—logic utilization becomes far less explainable by our model.

Normal estimation

In sharp contrast to the Histogram design space, Normal estimation is very well modelled. r^2 for logic and performance both increase towards the Pareto front. Gini coefficients are split, timing becoming more attributable to a subset of parameters, while logic becomes less so. Altogether this implies that the model parameters are of the same order of magnitude in importance and have proportional (or inversely proportional) relationships to the resulting performance and area.

While these two design spaces are extreme examples on the spectrum of nonlinearity they demonstrate that inspection alone is insufficient to determine the knob-to-design mapping. Figure 3.4 shows the model predictions alongside the true performance and area results. This example analysis shows that researchers should be very cautious with parametric models in DSE. Even very general techniques such as Gaussian process regression (see [130]) have hyperparameters that must be carefully tuned.

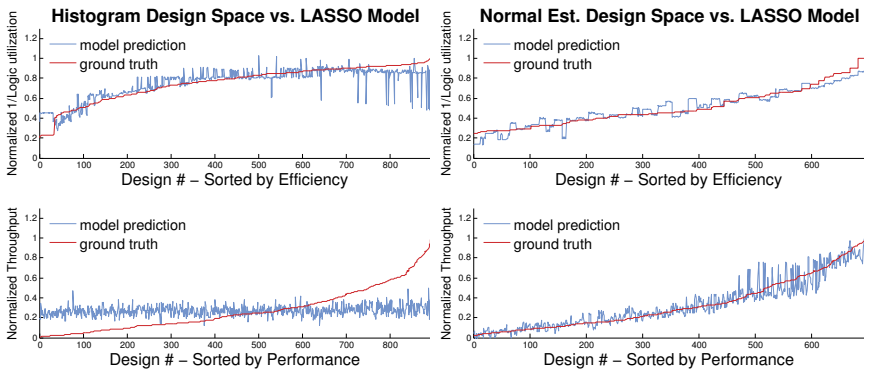


Figure 3.4. In the above figure we have sorted the ground truth designs and plotted them alongside the LASSO model predictions. The worst designs begin on the left and progress toward the best designs on the right. The Histogram model predicts performance very poorly, and while the logic utilization model appears to follow rather closely for mediocre designs, the most efficient designs are poorly modeled. The opposite is true for Normal estimation: Near-Pareto designs are modeled more accurately than the remainder of the space.

3.6 Conclusion

We have created a set of OpenCL benchmarks targeted at FPGA design space exploration for high-level synthesis. These benchmarks and the corresponding results can expand our knowledge on how to improve design choices. We have analyzed the results to show that the variations between designs can be affected not only by individual parameters, but also by complex interactions between these parameters that are difficult to model mathematically. Yet we have barely scratched the surface of the information that we can gather from these data, and further analysis is required to understand how to navigate design spaces efficiently. This work is a basis that can be leveraged by machine learning methods to refine optimization techniques.

Acknowledgements

This work was supported in part by an Amazon Web Services Research Education grant.

Chapter 3, in part, is a reprint of the material as it appears in the International Conference on Field-Programmable Technology (FPT) 2016. “Spector: An opencl fpga benchmark suite.” Gautier, Quentin; Althoff, Alric; Meng, Pingfan; and Kastner, Ryan. The dissertation author was

the primary investigator and author of this paper.

Chapter 4

FPGA Design Space Exploration With Hint Data

4.1 Introduction

High-level synthesis (HLS) has emerged as a powerful tool for increasing designer productivity and opening up the hardware design process to software programmers. Unfortunately, it is not a panacea. Designers still must expose parallelism and perform code refactoring to tailor the application towards the target hardware platform. Eking out the best design typically requires deep insight into both the application and the underlying hardware.

In order to optimize an application, a designer typically goes through a design space exploration (DSE) process where they parameterize that application, tune the parameters, synthesize to determine the performance and resource usage, and repeat the tuning process ad infinitum (or until they exhaust themselves or run up against the time-to-market constraint). Fully synthesizing a single FPGA design can take hours. And because the space of all possible configurations can contain hundreds, thousands, or even more designs, a brute-force optimization approach would take days to weeks of synthesis time. To address this DSE problem, several machine learning framework were developed to intelligently explore the space of possible designs [77, 74, 132]. These frameworks aim to sample the minimum amount of designs and find only the Pareto-optimal designs w.r.t. performance and resource utilization.

It is possible to get some "hints" about the quality of the design without going through the

entire synthesis process. For example, an HLS tool typically provides throughput and resource utilization estimates before the lengthy place-and-route process (i.e., *pre-PnR estimations*), which is by far the most time consuming aspect of synthesis. While these estimates provide useful global information, they are not entirely accurate and often fail to fully model the clock period, resource utilization, and other low level details that are greatly influenced by physical synthesis.

Taking this even further, one could also derive estimates by running the same code on other non-FPGA compute platforms. For example, when using the OpenCL language, the same code can be compiled to CPUs and GPUs. In this process, the functionality of the design is preserved across platforms, but the performance is not. Using OpenCL to compile the same designs on a GPU or a CPU is a very fast process, and thus we can often perform a brute force search across the entire space or at minimum evaluate orders of magnitude more designs compared to what we could do with FPGA synthesis. However, the underlying architectures are different and the results may not directly translate to the FPGA optimization. Yet, we show that it does provide useful nuggets of information that, if properly leveraged, can help the FPGA DSE process.

In this chapter, we explore how external information (hint data) can help HLS design space exploration. We are particularly interested in the case where the hint data do not come from a fine-tuned analytical model, but from data acquired easily and automatically. We use hint data from pre-PnR estimations in addition to CPU and GPU performance data. These data are estimates, and some are better than others, but they are all derived from the same design specifications. As such, it is possible to find subsets of the hint data that correlates with the FPGA data. We show how to leverage these potentially subtle correlations in order to guide the sampling process of FPGA design space exploration. In other words, we develop a framework that leverages external data as hints to better guide the machine learning algorithms in the DSE process. Our contributions are as follows:

- We provide a comprehensive framework for exploiting external data in the FPGA DSE

process, that is completely agnostic to the type of data.

- We compare several methods to extract useful information from inaccurate FPGA estimation data.
- We evaluate our methods on OpenCL FPGA benchmarks covering multiple types of applications and thousands of optimization combinations.
- We use three different types of hint data to improve the DSE overall efficiency up to 33%.

The remainder of this chapter is organized as follows. In Section 4.2, we describe the problem and present related work. We describe our baseline framework in Section 4.3. Section 4.4 gives an overview of our proposed methods, which are described in more details in Section 4.5. Section 4.6 presents our experimental protocol, and we show the results of our experiments in Section 4.7. Finally, we conclude in Section 4.8.

4.2 Design Space Exploration (DSE) for FPGAs

4.2.1 Definitions

When performing design space exploration, we must carefully understand the application that we want to optimize and develop a set of parameters that modify the design in order to provide better performance and/or resource usage. For example, we may have a parameter that states how many times to unroll a `for` loop. Each modification is represented as a parameter, which we call a *knob*.

Knobs can be *categorical*, *continuous* or *discrete*. Categorical knobs can take multiple non-numerical values to represent a decision (e.g., a boolean enabling/disabling a hardware path, or switching between multiple possible implementations); continuous and discrete knobs take a numerical value that has a direct impact on the design (e.g., unroll factor, initiation interval, etc.). In this paper, we only consider discrete knobs by assigning values to categorical knobs and discretizing continuous knobs with hardware-friendly values (e.g., powers of 2).

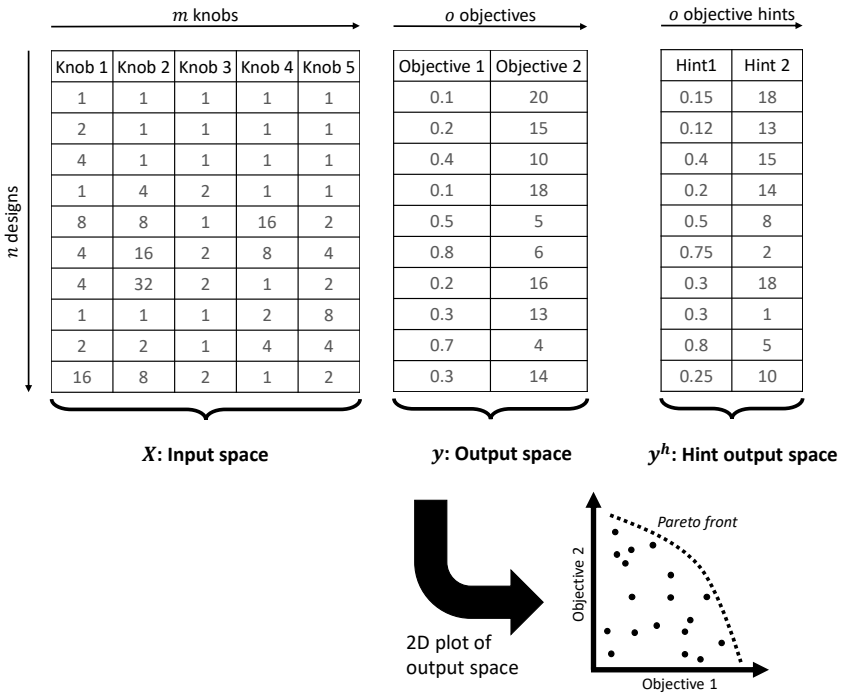


Figure 4.1. Illustration of the concepts of input space and output space that both form a design space.

We define the space of all valid combinations of knobs, by considering the conditions and constraints that may exist between the knobs, where each combination yields one possible *design* for the application. This set of knob values and all possible knob combinations forms the *input space*, represented by a matrix X of dimensions $(n \times m)$, where n is the number of knob combinations (i.e., the number of design candidates), and m is the number of knobs. Each row of X represents a unique combination of knob values, and thus, a unique design. Figure 4.1 illustrates the input space matrix X.

The goal is then to *explore* this space. We define *exploring* as maximizing or minimizing a number of *objectives* that are given by the evaluation of each possible design. In the case of hardware applications, the two objectives that we are optimizing are 1) throughput and 2) resource utilization. Ground truth results for these objectives are obtained by fully synthesizing and implementing designs from standard benchmark suites (e.g., [98, 42]) to an FPGA, and evaluating them on a dataset representative of the target application. These results form the

output space (see Figure 4.1), represented by a matrix y of dimensions $(n \times o)$, where o is the number of objectives. Each row of y is a fully resolved sample. In practice, ground truth data are typically not available; in other words, y is unknown. However, we assume that there exists a relationship between X and y, i.e., there exists a function $f : \mathbf{R}^m \rightarrow \mathbf{R}^o$ such that:

$$y = f(X)$$

We assume that f can potentially be non-linear, and thus would require a large number of samples to be estimated. Furthermore, since the rows of y generally come from measurements (e.g., running time, accuracy, etc.), we consider that the output space is noisy, which means that a proper design space exploration tool needs to be robust to noise in order to build a stable model of f. A *design space* is the combination of an input space and an output space, although the designation is often informally used to refer to either the input space, output space, or both.

4.2.2 Methods

There are multiple ways to explore the design space and DSE is broadly divided into *synthesis-based* methods and *predictive-based* methods, and a few hybrid solutions. Synthesis-based methods [77, 74, 132, 100] rely on machine learning to predict the entire design space by sampling a small number of implementations, synthesizing them, and measuring the actual results of the design. These methods effectively reduce the compute time compared to a brute-force approach. However, these works only rely on data from the target architecture (FPGA) and overlook the opportunity to use training data from another source, which may provide a further reduction of the DSE time. Predictive-based methods [75, 117, 99, 127, 126, 115] create an analytical or probabilistic model based on the target architecture, and potentially analyze the source code directly to estimate the final performance and required resources. The created models can be used to predict or estimate the design space, accelerating its exploration since it does not require any synthesis. Predictive models depend a lot on the chosen training set and

on parameter tuning in order to achieve the best accuracy. Hybrids methods aim to combine both prediction and synthesis, e.g., Liu and Schafer [73] use the pre-place-and-route resource estimates to guide the sampling process. Another example is the work of Cui et al [28] who build predictive models for circuit delay and use transfer learning from these models to improve the prediction of delay on true circuits.

Our work presents similar ideas as the last two examples, but generalizes the concepts to multiple types and sources of hint data. We do not only rely on pre-PnR estimations given by the tool. Our methods are more general, as we show how to use any data to guide the DSE process. We use these data to guide a full iterative active learning process aimed at finding and sampling the Pareto-optimal designs. Additionally, we consider the HLS tool and the target FPGA architecture as a black box for which we do not have - and thus do not need - any information. In essence, we create a hybrid method using a performance model to help the synthesis-based sampling, where the performance model is general and does not need to be based on the target architecture.

4.3 ATNE Sampling

This work is based upon an existing design space exploration framework called Adaptive Threshold Non-pareto Elimination (ATNE) [77]. ATNE is a machine learning method that searches for *optimal* designs in a pool of candidates while minimizing the number of designs that need to be sampled. Designs are considered optimal based on *objectives* specified by the designer, and representative of the goal of the application. In hardware design, objectives are often related in a way that makes it impossible to perfectly optimize all the objectives simultaneously. Typically, improving throughput requires to increase the logic utilization and conversely. This is why many DSE algorithms – ATNE included – aim to find designs that are *Pareto-optimal* with respect to the user-defined objectives. In other words, we are trying to find the Pareto front of the design space. The Pareto front represents the set of designs within the

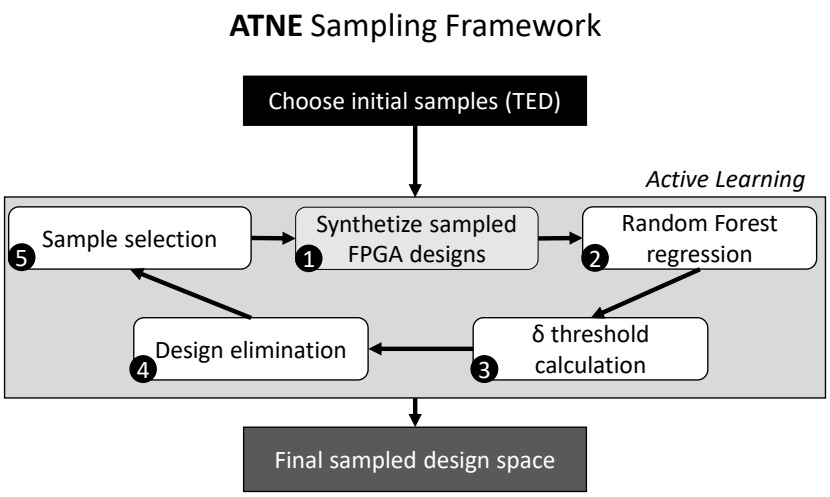


Figure 4.2. Overview of the original ATNE sampling framework. ATNE is composed of an initial sampling step performed by the TED algorithm, followed by an iterative active learning process.

design space that cannot be improved further on one objective without making another objective worse. An example of design spaces with two objectives and highlighted Pareto-optimal designs can be seen in Figure 4.4 and 4.5.

The original ATNE algorithm optimizes for throughput and logic utilization. Our open-source implementation of ATNE (see Section 4.6) can handle any number of objectives, although we focus on two objectives in the remainder of this paper.

ATNE employs active learning [102], which is a process where a new data label is queried at each iteration to improve the model. ATNE enhances this process by focusing the queries into the optimal portion of the output space by regularly *eliminating* non-optimal design candidates. Figure 4.2 presents the main steps of the process that we briefly describe below. The original paper [77] contains the relevant pseudo-code and equations.

4.3.1 Initial Sampling

The algorithms starts by sampling an initial number of designs. This initial set of samples is built using the Transductive Experimental Design (TED) algorithm [121]. TED is a sampling method by itself, however it is not meant to produce samples matching the ground truth Pareto

front. It is meant to choose a representative set of experiments that we can use to estimate the global design space. TED only operates on the input space (knob values), and as such does not benefit from valuable feedback from the output space, but it is a good choice as an initial step to build a robust regression model.

4.3.2 Regression Model

Figure 4.2, steps 1 and 2. The initial set of selected samples is fully synthesized, compiled, and run on FPGA to measure the output objective values (e.g., throughput and utilization). We call this set, the set of *known* designs. The known outputs are then used to build a regression model to estimate the shape of the entire space. ATNE uses a random forest regressor [15] that uses the inputs (knob values) and outputs (measured objectives) of the known designs, and estimates the outputs of the *unknown* designs. One of the key components of ATNE is that it actually builds several models from different subsets of the known data, based on a bootstrap ratio. It uses n_f random forests, each with n_t trees, and each training on the known designs to create a model and predict the unknown data. Thus, each unknown design has multiple (n_f) estimated output values.

4.3.3 Design Elimination

Figure 4.2, steps 3 and 4. Each random forest produces one set of predicted output values. The algorithm calculates a pairwise subtraction of all these values on each set, and computes the standard deviation of each resulting subtraction across the sets. The subtraction creates an order between all pairs of designs (i.e., which design is better, and by what amount), and the standard deviation represents the divergence between multiple models. This standard deviation on each value is used to derive a threshold δ that is used as a margin of error in order to eliminate non-optimal designs. δ is controlled by a user-defined parameter α, which can be tuned to make the elimination step more or less aggressive. The idea is to eliminate designs that are dominated in most models. However, using a strict domination criterion would lead to a high

number of false negatives. Therefore, the algorithm only eliminates designs that are dominated by a distance δ calculated independently on each objective. Once identified, the potentially non-optimal designs are simply removed from the set of unknown designs and not considered in subsequent iterations.

4.3.4 Active Sampling

Figure 4.2, steps 5 and 1. Finally, the algorithm calculates a score for each remaining unknown design based on its dominance over other designs across all the regression models. This score gets higher as a design is estimated Pareto-optimal by a large number of models. The design with the highest score is chosen, synthesized and run, then added to the set of known designs. In the next and all subsequent iterations, the regression models are re-computed with the newly sampled data, and all the previous steps are applied until a sampling budget is reached, or until there are no more remaining design candidates.

4.4 Hint Data Overview

FPGA design data are very time-consuming to sample (hours), however we can easily collect other, related types of data quickly (seconds or minutes), e.g., estimations of the design tools, executing the code on other architectures, etc. These data can generally not be used directly to guess the performance of FPGA designs, but they can be used as a hint to help finding the best designs and eliminating the non-optimal ones. We want a general design space exploration technique that can take as input any arbitrary data as a hint and use this information to guide the exploration process. The key is extracting the useful data from the external hint data in a generic manner that is robust to noise and to misleading information.

Our goal is to investigate several methods of various complexity to extract information from hint data. We want to design and test these methods on multiple datasets in order to quantify the improvement (or deterioration) of the quality of results when applying these methods to a synthesis-based DSE algorithm.

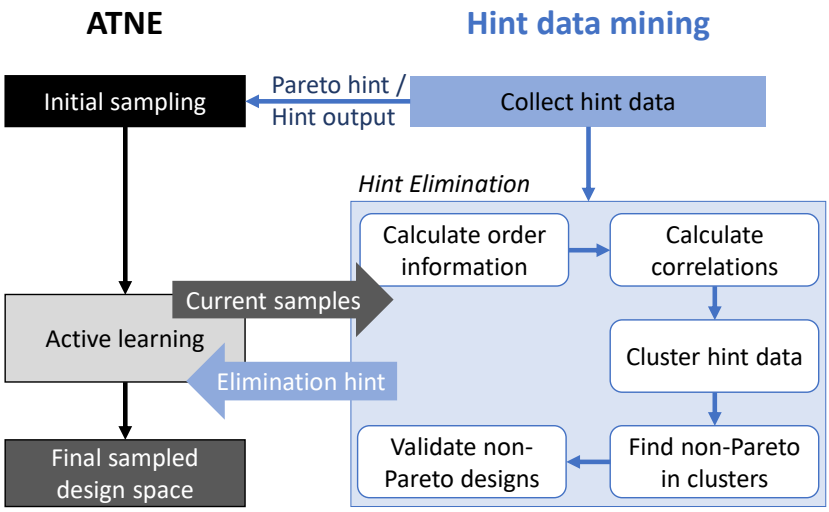


Figure 4.3. Overview of our proposed methods to improve the sampling results. We propose to improve the initial sampling with information from the hint data, and we design a hint elimination algorithm to further help the elimination step of ATNE.

We extend the ATNE algorithm presented in Section 4.3 and the TED algorithm that is used as initialization of ATNE. We enhance them with both a simple and a more complex technique to extract and use external *hint* information as illustrated in Figure 4.3. We measure the effectiveness of our hint-based techniques by measuring the total number of designs evaluated (*sampled*), and by calculating an error metric. We use the Average Distance to Reference Set (ADRS) metric [92, 94]. ADRS measures the average normalized distance between the estimated Pareto front and the ground truth Pareto front. As it is a normalized value, we report it as a percentage. The closer it is to 0, the better the estimation is.

There exists multiple ways to extract useful information from data coming from external sources. Here we define the concept of *hint* data and the assumptions that we make about these data. We consider as hint data, any data that can be obtained at least an order of magnitude faster than *post*-PnR (place-and-route) FPGA results, and that likely have some level of correlation with FPGA data. This includes pre-PnR estimations, GPU or CPU runs of OpenCL designs, or data coming from an analytical model. A hint design space contains the same input space X as the FPGA design space, but contains a different output space represented by the matrix y^h

of dimensions $(n \times o)$ (see Figure 4.1). As opposed to y, we consider that the values of y^h are known (as they are very fast to obtain). We also assume that we have data on all target objectives so that the hint output space can form a design space of the same dimensionality as the FPGA output space. In our case, we combine area utilization estimates from the pre-PnR results as the first hint objective, with performance data from either pre-PnR results, GPU or CPU runs as the second objective. We create hint output spaces y^h_{est}, y^h_{GPU}, and y^h_{CPU}. Note that, technically, the hint data do not need to be complete, i.e., y^h can be incomplete, although we only experimented with complete hint spaces in this paper. Finally, while we assume that some subsets of y^h are correlated to subsets of y, we also assume that y^h is noisy and thus not completely reliable.

We can extract information from these hint design spaces and use them to improve either the sampling algorithm directly, the set of initial samples, or both (see Figure 4.3).

4.5 Exploiting Hint Data

We aim to most effectively utilize the hint data in order to make our FPGA design space exploration faster and more accurate. In this section, we discuss various techniques to extract useful information from hint data to improve the DSE algorithm. We divide our techniques in three categories: A) techniques using the hint data directly without running the DSE algorithm, B) techniques improving the initialization of the DSE algorithm, and C) techniques to improve the elimination step of the DSE algorithm.

4.5.1 Directly Using Hint Data

The best scenario is when the hint space correlates extremely well with the FPGA design space. In this case, we could quickly find the Pareto-optimal designs in the hint space and directly use these as the best designs in the FPGA space. This eliminates costly FPGA synthesis and if accurate would result in a very fast design space exploration process. But this is all contingent on how well these two design spaces track one another.

Figure 4.4 shows the hint and FPGA output spaces (for the same input space) for the

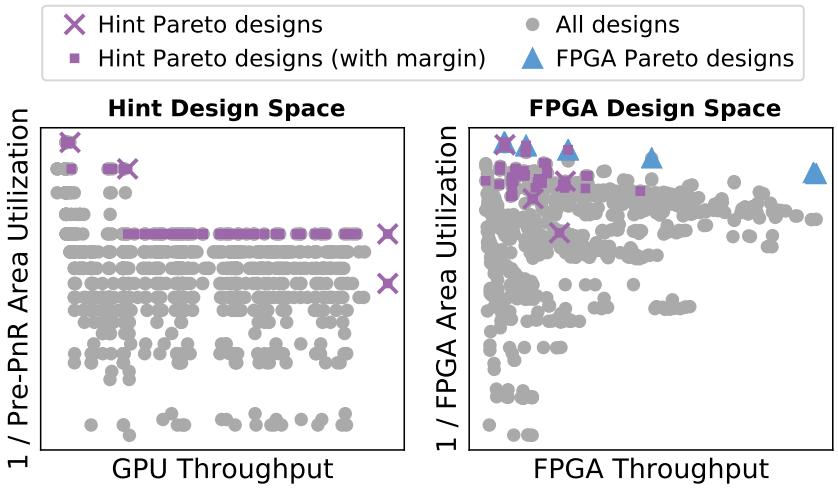


Figure 4.4. Example of directly using the hint data to find the Pareto-optimal points. On the left is the hint space from pre-PnR resource data and GPU performance data. On the right is the ground truth FPGA design space. Strict optimal designs (crosses) and optimal design with a 1% margin (squares) from the hint space are plotted in the FPGA space. They are very different from the true optimal set (triangles).

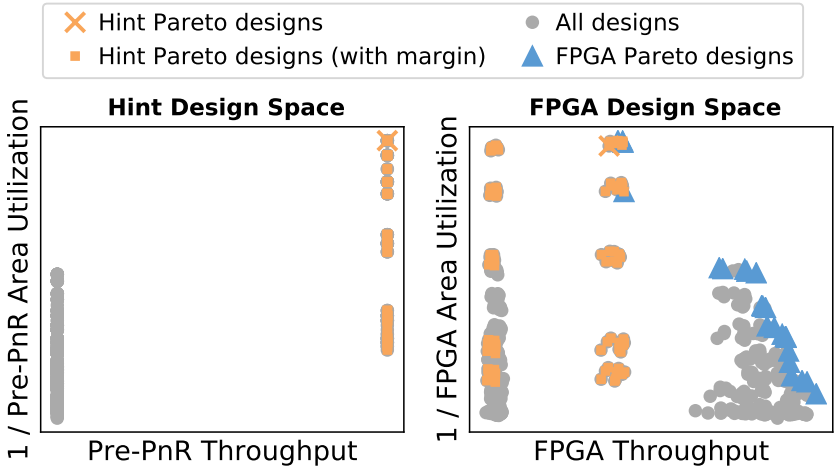


Figure 4.5. Directly using the hint Pareto-optimal designs on a different benchmark, using the pre-PnR throughput estimations. Selecting optimal designs from the hint space with a margin of 1% leads to sampling 90 designs.

histogram benchmark from our dataset (see Section 4.6.3). The left graph plots the hint design space; it uses GPU results for throughput and pre-place and route (pre-PnR) results for utilization. The FPGA design space on the right corresponds to the fully synthesized and implemented FPGA results for throughput and utilization (obtained from an estimated 2200 hours of computation).

On the hint space, the Pareto-optimal designs are highlighted with crosses, and on the FPGA space, the optimal designs are highlighted with triangles.

A quick observation of the plots shows that the design spaces do not look similar. For example, the FPGA design space has many designs clustered in the upper left corner (low utilization and low throughput), and its Pareto front has a low gradient curve. The hint design space is more evenly distributed and the slope of the Pareto front curve is steeper.

However, the shape of the entire design space is not as important as whether the Pareto optimal designs are correlated. We do not care much about the worst and non-optimal designs. If the best (Pareto-optimal) designs correlate well, then we can still directly utilize the hint space to find the Pareto designs in the FPGA space.

We calculated the Pareto-optimal designs in the hint space and highlighted these designs in the ground truth FPGA design space. For comparison, we also calculated the Pareto optimal designs in the FPGA design space. We can see that the estimated Pareto designs (crosses) are far from the ground truth Pareto (triangles).

This observation holds when we relax the condition of optimality by selecting designs within a certain distance of the Pareto front. Here we define a Pareto margin of 1% of the distance between the strict Pareto front and the minimum value on each axis. On Figure 4.4 and 4.5, the designs within this margin are highlighted using small squares on the hint space, and reported on the FPGA space for comparison. This margin is insufficient to attain the true optimal designs, while significantly increasing the number of selected samples. The second example in Figure 4.5 (*BFS dense* benchmark) shows that even a 1% margin can lead to a very large number of sampled designs; in this case 90 designs are selected on the hint space, and are all far from the FPGA Pareto front when plotted into the FPGA space. We observe that the estimated optimal designs are generally better on the area utilization axis. This reflects the ability of the synthesis tool to properly estimate the hardware architecture. While this method can sometimes work when hint data are very correlated, its results are difficult to predict and vary greatly between design spaces (see Section 4.7 for more detailed results). We need to implement methods that can produce a

more reliable outcome.

4.5.2 Improving initialization

The choice of initialization algorithm for learning-based DSE methods can have a large impact on the quality of results. A smart selection of designs representative of the entire space is better than a randomized selection [74]. Transductive Experimental Design (TED) [121] selects a representative subset of experiments (i.e., points on the input space) to better characterize the output space. For our problem, this means that TED selects a set of initial designs to synthesize before starting the machine learning process. TED does not make any assumptions on the learning model and is therefore suited to start the active learning stage of ATNE. We want to investigate methods to improve the design sampling performed by the TED algorithm by using our collected hint data.

We use *domain adaptation* [95] to improve our information extraction process. The exact definition of domain adaptation sometimes varies; we use it as a subset of transfer learning where we attempt to adapt a model created in a source domain to predict labels in a target domain. In our case, the hint design space is the source domain and the FPGA design space is the target domain. As the hint output space is fully resolved, we can use it as a base model to adapt to the target space.

The first use of domain adaptation technique applies information from the source domain to initialize the hypothesis in the target domain. In our case, we use the hint space to improve the selection of initial samples in the target space. We want to improve TED with information from the hint data. In the following, we describe and compare three different methods to achieve this goal.

Pareto Hint + TED ($\mathbf{TED_{hpar}}$)

Our first method uses the Pareto-optimal hint designs to initialize the sampling framework. The goal of the sampling process is to target the Pareto front. Therefore, if the Pareto hint designs

are similar to the target Pareto designs, the sampling algorithm will provide an initial bias toward the Pareto front.

The size of the strict Pareto front is typically limited to less than ten designs. In the case where there are not enough Pareto hint designs to meet the initial sampling budget, we have two choices: increase the Pareto margin or initialize the remaining designs with TED. The problem with increasing the Pareto margin is that it produces unstable results (see Figure 4.5). Additionally, we assume that TED is generally better at selecting useful designs to build a model. Therefore we use TED to initialize at least half of the initial sampling budget, along with the optimal designs from hint data.

TED on Hint Output Space ($\mathbf{TED_{out}}$)

The goal of TED is to select experiments that are likely to be representative of the entire output space, but without knowledge of the outcome of these experiments. In other words, TED operates on the input space only. Basically, TED provides an even distribution of samples in the input space without creating dense clusters. In our case, the algorithm considers the knob values and tries to create a grid-like distribution of samples in this space. Since the hint output space is fully resolved (i.e., we have hint values on all the objectives for each design), we can take advantage of that additional information by asking TED to create an even distribution of samples in the output space. We are more interested by the shape of the output space (i.e., what are the values of the objectives for each design on FPGA) and therefore, by selecting a representative subset of the hint output space, we can hope that it will also be representative of the FPGA output space.

TED on All Hint Output Spaces ($\mathbf{TED_{all}}$)

Running TED on one output space (TED_{out}) focuses the algorithm on finding evenly spaced samples in that particular space. However, one particular hint space might be too noisy or biased in a certain way. Since we can potentially obtain performance data from multiple

sources (CPU, GPU, models, etc.), we can leverage all this information to help the sampling. We concatenate the matrices of multiple output spaces to create a new, enhanced output space. Essentially, we add extra dimensions to our output data, which forces TED to evenly distribute points in these new dimensions as well. This new distribution has the potential to include samples with high disagreement among hint data, which can help reduce noise from these hints. Additionally, given enough initial samples, TED will create a representative subset of each hint space, which increases the probability of selecting a good representative subset of the FPGA space.

4.5.3 Improving Elimination With Hint Data

A fundamental idea of the ATNE algorithm is to progressively eliminate non-optimal samples from the set of design candidates. The elimination step of the algorithm removes designs from consideration if it is reasonably confident that they are not Pareto-optimal. Domain adaptation techniques can utilize source domain information to further improve target domain learning. Combining these two concepts, we can leverage similarities between the hint design space and the FPGA design space in order to eliminate more designs. We present a method to "mine" the hint data for relevant information about the optimality of FPGA designs, and effectively use that information to discard a subset of the design space. We can perform this process at any time during the sampling algorithm. In our experiments, we run it at each iteration of the ATNE active learning loop.

Hint Elimination Algorithm

This algorithm takes as input the hint data along with the sampled FPGA data, finds useful *information* in these data, process it, and return a suggested set of designs that are likely non-optimal. These designs can then be eliminated by the ATNE algorithm to avoid sampling them and thus accelerate the overall convergence.

The hint data are provided on all objectives and create a fully resolved design space

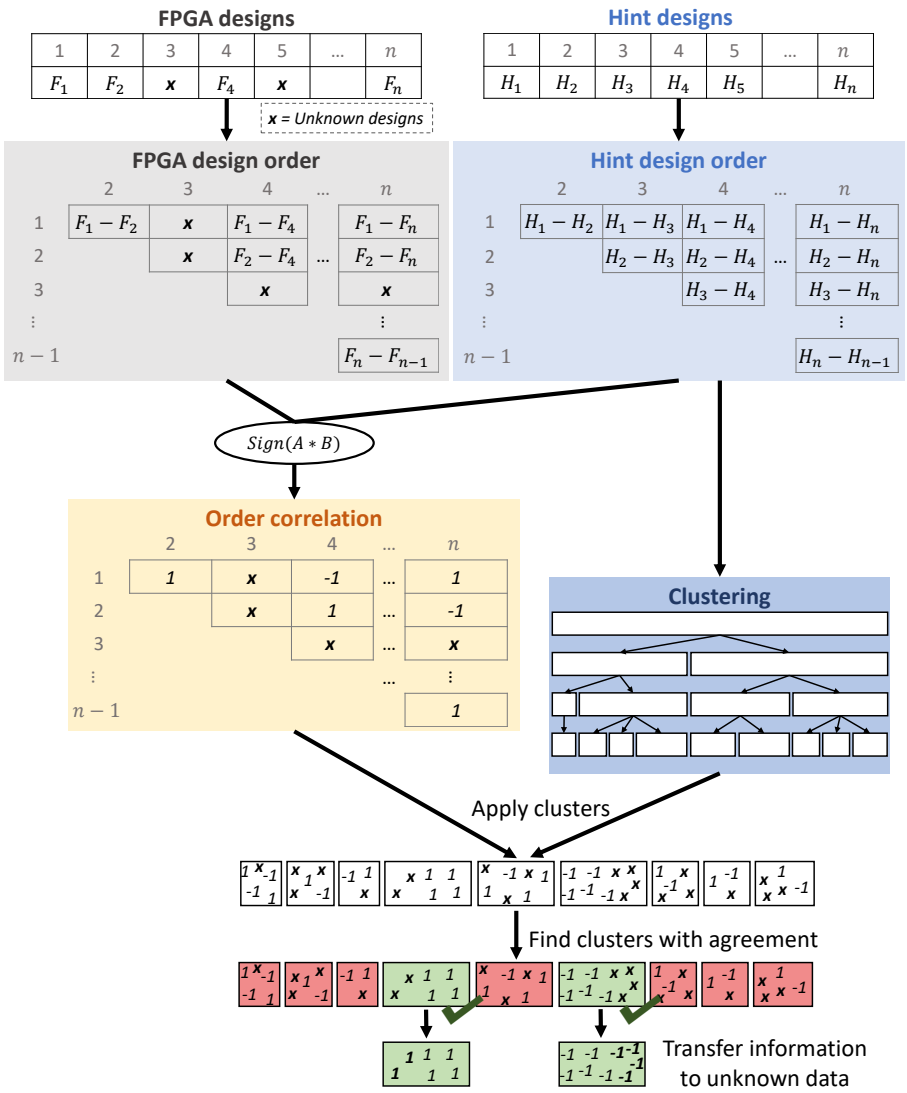


Figure 4.6. Our hint elimination method. 1) Calculate matrix of design order (pairwise differences between designs on one objective) for known FPGA designs and hint designs. 2) Calculate a simplified correlation between the two. 3) Cluster the hint design order. 4) Apply these clusters to the correlation data. 5) Find clusters containing only the same values and propagate that value to the unknown FPGA data.

y^h. Since we assume that y^h contains a certain level of noise, we need to design a method to discriminate between useful information and information that could hurt the performance of the algorithm (e.g., by suggesting to eliminate designs that are in fact Pareto-optimal).

The information that we are looking for in this method is the *design order*. Considering two designs A and B, for one particular objective, we need to determine if design A outperforms

design B, or the opposite. We define it simply by the subtraction

$$Order = A - B$$

where $Order$, A and B are vectors of o elements (the number of objectives). In this method, we are particularly interested in the sign of this equation that represents the relative ordering (better/worse) between designs on each axis.

We want to calculate and compare *design order* in both the hint space and the FPGA space. In particular, we want to find clusters of information where hint design order and FPGA design order are in agreement. In such clusters, we infer that unknown FPGA designs follow the same trend. We can then select elimination candidates based on these information. The accuracy of our predictions depends on selecting the proper size for clusters. We provide more details on cluster determination below.

Figure 4.6 illustrates the processed applied to each DSE objective. The first step of the process is to calculate the design order of the known FPGA designs and the hint data. For each design space, we calculate a matrix of pairwise differences, and only keep the upper half to avoid redundant information.

The second step is to compute a simplified correlation: 1 if both the hint order and the FPGA order agree, -1 if they disagree, and *unknown* if the value is unknown on FPGA data. At this point, we have a matrix composed of 1 and -1 that is hard to exploit.

Our third step is to find clusters on the hint order matrix, and apply these clusters to the correlation matrix. The result is a number of clusters containing 1, -1 and *unknown* values. Within each cluster, we look at the known values. If some of these clusters only contain perfectly agreeing values, i.e. contain only 1 or only -1, we consider that the unknown values are similar. From this, we can deduct that within such a cluster, all the orders are correlated or anti-correlated.

Using the information from these agreeing clusters, we can compute which unknown designs are sub-optimal on all objectives, and mark them as candidates for elimination. The last

step not illustrated here is the validation of the elimination candidates. This step is currently tightly integrated with ATNE, but could easily be adapted to a different framework. We use the random forest regressors to verify that the prediction model that they build agrees that the designs are non-optimal. We do not check for perfect agreement, otherwise it would indicate that the original algorithm can already find non-optimal designs, but we look for very large disagreement to remove potential candidates for which the uncertainty is very high. In other words, if the disagreement is too high, we safely decide to keep that design for another iteration.

The entire process is described in details in Algorithm 3. The first top-level loop corresponds to the creation of the order matrices and the correlation matrix, and the clustering of the hint data. The clustering method is detailed below and in Algorithm 4. The second loop shows how designs are chosen for elimination from the cluster information. Finally the third loop presents the validation process. Note that our algorithm is agnostic to the type and number of objectives that it can work on. We can potentially run it on a design space exploration that would include a third axis such as an accuracy metric for example.

Clustering the Hint Data

One important feature of our method is to be able to cluster the hint data in order to infer unknown FPGA data. But in order to get the best accuracy in our predictions, we have to choose the cluster size carefully. More specifically, we need to determine how many known values must exist inside a cluster to maximize the probability of predicting correctly. The number of known values that we need is a threshold T. If T is too small, then we risk using too few known values to predict unknown values. But if T is too large, we risk that the clusters are not meaningful as they can contain heterogeneous data. Therefore it is important to choose a good threshold. To that end, we formalize the problem and provide a theoretical solution.

Our goal is to find clusters such that the simplified correlation matrix composed of 1 and -1 can be used to predict whether unknown values are 1 or -1. The sampling process within each group can be mathematically modeled as drawing boolean samples without replacement.

Algorithm 3: Hint Data Mining Algorithm

Input : Hint data; Sampled FPGA data; Predicted Space by regressors

for $o = 1$ *to* $N_{objectives}$ **do**
  $Order_{FPGA}(o)$ = pairwise differences for FPGA data
  $Order_{HINT}(o)$ = pairwise differences for Hint data
  Initialize $Corr(o)$ matrix to 0
  for *each pair* (f,h) *in* $(Order_{FPGA}(o), Order_{HINT}(o))$ **do**
    c = 0
    if f *is known* **then**
      if $(f * h) > 0$ **then** c = 1;
      else if $(f * h) < 0$ **then** c = -1;
    end
    Save c into $Corr(o)$
  end
  $(clusterIndexes(o), clusterValues(o)) = ClusterRecursive(Order_{HINT}(o), Corr(o))$
end
EliminationCandidates = $\emptyset$
for $i = 1$ *to* $N_{designs}$ **do**
  for $j = 1$ *to* $N_{designs}$ **do**
    for $o = 1$ *to* $N_{objectives}$ **do**
      cidx = $clusterIndexes(o, i, j)$
      corr = 0
      if *all* $clusterValues(o, cidx) == 1$ **then** corr = 1;
      else if *all* $clusterValues(o, cidx)$ == *-1* **then** corr = -1;
      $Order_{FPGA}$(o,i,j) = corr * $Order_{HINT}$(o,i,j)
    end
    if $Order_{FPGA}(i, j) < 0$ *for all objectives* **then**
      Add i to EliminationCandidates
    else if $Order_{FPGA}(i, j) > 0$ *for all objectives* **then**
      Add j to EliminationCandidates
    end
  end
end
for *each c in EliminationCandidates* **do**
  differences = $\emptyset$
  for *each design in* $P_{relaxed}$ **do**
    d = $predictedValues(design) - predictedValues(c)$
    Add d to differences
  end
  $N_{inferior}$ = Num designs s.t. $differences > 0$ for all objectives
  if $N_{inferior} < 5$ **then** Remove c from EliminationCandidates;
end

Output : EliminationCandidates

Algorithm 4: Recursive Clustering

```
Procedure ClusterRecursive(Order_HINT, Corr)
    Compute sampling threshold T using Equation 4.2
    if Num known values in Corr > T and Corr contains both 1 and −1 then
        Idx_L, Idx_R = Cluster Order_HINT into 2 groups
        CIdx_L, Values_L = ClusterRecursive(Order_HINT(Idx_L), Corr(Idx_L))
        CIdx_R, Values_R = ClusterRecursive(Order_HINT(Idx_R), Corr(Idx_R))
        CIdx = Combine CIdx_L and CIdx_R
        Values = Combine Values_L and Values_R
    else
        CIdx = matrix of all 1
        Values(1) = Corr
        if Num known values in Corr < T then
            Values(1) = all 0
        end
    end
    Return CIdx, Values
```

Therefore, its underlying probability distribution is hypergeometric:

$$P(X = k) = \frac{C_k^K C_{n-k}^{N-K}}{C_n^N} \tag{4.1}$$

where C_j^i is "i choose j", $N =$ the population size, $K =$ the number of 1's versus -1's in the population, $k =$ the number of 1s drawn, and $n =$ the total number of samples.

Under this model the ideal threshold n is the answer to the question, "With a fixed likelihood $1-\beta$, what is the minimum number of samples n for which I am likely with probability $1-\beta$ to draw all 1's even though the population contains (εN) -1's and $(1-\varepsilon N)$ 1's?" To see this more clearly, note that we are attempting to determine how likely it is that we have been tricked by our sampling limitations into thinking that a cluster is either entirely composed of 1's, or vice versa. We use a recursive numerical method described in equation 4.2 to solve this problem.

$$f(x) := \left\{ \begin{array}{ll} x & \text{if } \frac{N!(\hat{K}-x)!}{\hat{K}!(N-x)!} \leq \beta \text{ or } x = N \\ f(x+1) & \text{otherwise} \end{array} \right\} \tag{4.2}$$

where $\hat{K} = \lfloor N(1-\varepsilon) \rceil$. We let the user of our tool provide β and ε as two parameters describing

how much error is acceptable for the DSE. Then, we can take the β and ε into the numerical method to produce the appropriate T.

The equation to calculate T is included in the clustering algorithm presented in Algorithm 4. This algorithm is recursive and works as follows. First it considers four configurations: 1) The cluster contains exactly T known values, it's a termination condition 2) The cluster contains less than T known values, it is marked as invalid and terminates 3) The cluster is *pure*, the algorithm terminate 4) There are more than T unknown values and the cluster is not *pure*. In the fourth configuration, the hint data are clustered into two groups. Then each group is clustered recursively, the results are combined together and the algorithm terminates.

4.6 Experimental Setup

In order to test the validity and performance of our methods, we implement and run these methods on multiple datasets, and compare the results with a baseline algorithm. In this section, we present the algorithms that we use and how we tune them, along with the datasets we use, and the metrics that we use and define to measure the outcome of our experiments.

4.6.1 Algorithms

In our experiments, we use the Adaptive Threshold Non-Pareto Elimination (ATNE) algorithm from [77] as a baseline DSE method, and we extend it to include the various hint-based exploration methods described in Section 4.5. We implement ATNE using Python 3 and the Random Forest implementation from the *scikit-learn* package. All the source code for our implementation of ATNE extended with our methods is made available open-source [1]. ATNE uses the Transductive Experimental Design (TED) algorithm as an initialization process. We also use our Python implementation of this algorithm to test our enhanced initialization methods. ATNE outputs two values by default: The first is the total number of designs sampled by the algorithm after it converges (i.e., all possible designs have been sampled or eliminated). The second is the ADRS metric computed from the output set of Pareto designs and the set of ground

Table 4.1. ATNE and hint elimination parameters

Init Samples	Forests	Trees	Bootstrap	α	β	ε
15 / 40	30	200	0.5	0.999 / 0.2	0.1	0.25

truth Pareto designs. We also output the ADRS value at each iteration to observe the rate at which the error decreases, although this metric is difficult to summarize as explained in Section 4.6.5.

4.6.2 Hyperparameters

The ATNE algorithm has several parameters that can be tuned (Section 4.3): number of initial samples, number of random forests, number of trees per forest, forest bootstrap ratio, and the parameter α that controls how aggressive the algorithm is (i.e., how much the process eliminates designs, at the potential expense of accuracy). In order to tune these parameters, we generate a series of synthetic design spaces. The synthetic design spaces are created from random non-linear function with multi-dimensional input and two outputs. Each of these spaces have 200 designs. We combine nine synthetic designs spaces, with the smallest benchmark from our chosen benchmark suite (which has about 200 designs as well), and we perform a grid search of the hyperparameters on multiple independent subsets the set of spaces. We also search for the hyperparameters controlling the cluster size of our hint elimination method: β and ε. We pick values with an ADRS below 1%, and fix most of these values for our experiments. The two exceptions are: the number of initial samples, and the α parameter. The α parameter is at the core of the original ATNE algorithm, therefore we want to verify that our method works with a very different value (we pick 0.2, which is an extreme value considering that this parameter is very sensitive to small changes). The number of initial samples can influence the total number of selected designs, and is particularly important in several of our methods relying on modifying TED. For this reason, we chose to run experiments with both a small number of initial samples (15) and a higher number (40). All the values that we selected for all the parameters are summarized in Table 4.1.

4.6.3 Benchmarks

We use the Spector benchmark suite [42] to obtain ground truth data from FPGA implementations. Spector provides the input matrix X for each design, along with a ground truth output space y that we can use to measure the performance of our design space exploration methods. Our hyperparameter tuning process employs the *DCT* benchmark in addition to the synthetic design spaces; as a result, we do not include DCT in our experiment results.

4.6.4 Hint data

In addition to the Spector data, we have collected the pre-PnR estimation values from the Intel FPGA synthesis tool (same version as used in the benchmarks) for all the designs. We have also run the OpenCL designs on a Nvidia Tesla K20c GPU and an Intel i7-4790K CPU, and collected the throughput data. These three types of data can be combined to create several hint output space matrices y^h. For each hint output space, we create two axes: one axis for logic utilization hint, and one axis for throughput hint. The logic hint axis consists of pre-PnR resource utilization estimations from the synthesis tool. The throughput hint axis can be either either the pre-PnR throughput estimations from the synthesis tool (**Est.T.**), GPU throughput (**GPU**), or CPU throughput (**CPU**). Pre-PnR throughput estimations are often not reliable as they do not take into account data-dependent information such as external memory accesses, inter-modules communication and branches on the architecture. GPU and CPU data are not very reliable either due to architecture differences and some hardware knobs not having any effect on GPU/CPU, however, they have the advantage of measuring the memory accesses and all the data flow in general. Additionally, GPUs share some features of FPGAs such as some level of parallelism and a similar memory hierarchy.

4.6.5 Metrics

We run multiple versions of ATNE (modified with our methods and unmodified) with the parameters described above on all of the benchmarks, and take the average of 15 runs to account

for the non-determinism of the algorithm. We run ATNE until it converges to a solution, and collect two sets of metrics. The first set of metrics is composed of: 1) the sampling complexity (percentage of design space sampled), and 2) the ADRS (error between the sampled space and the ground truth). We also derive a third metric to summarize the first two, called *difficulty*. It is defined as the following: $\text{Difficulty} = (\text{Complexity} + \text{ADRS})/2$, and represents the global difficulty of learning a design space. As both the complexity and the ADRS are defined between 0 and 1, the difficulty gives an equal weight to the two metrics. For some specific applications, it can be useful to define the difficulty with different weights, but we present the general case here. We want to minimize all of these metrics, and compare the results of our methods (*target*) to the results of the original ATNE algorithm (*baseline*). For a fair comparison across benchmarks, we choose to normalize the results. However, in order to achieve a symmetrical scale between positive results (target is better) and negative results (baseline is better), we calculate a piecewise normalization:

$$\text{Improvement} = \left\{ \begin{array}{ll} \frac{\text{Baseline}-\text{Target}}{\text{Baseline}} & \text{if Baseline} > \text{Target} \\ \frac{\text{Baseline}-\text{Target}}{\text{Target}} & \text{if Target} > \text{Baseline} \\ 0 & \text{if Baseline} = \text{Target} \end{array} \right\} \tag{4.3}$$

This equation provides a scale defined as: 0% if the target and baseline results are equal; +100% if the target is perfect (equals 0) and the baseline is worse; -100% if the baseline was perfect and the target results are worse.

The second set of metrics is collected by calculating the ADRS relative to the ground truth at each iteration of the algorithm, which results in a curve describing how fast the algorithm is able to decrease the error and describe the true Pareto front (see Figures 4.11 and 4.12). However, because ATNE is eliminating samples, it eventually converges and stops sampling before reaching the sample budget (which we set to the total number of designs), and the convergence rate depends on the state of the random number generator. This situation results in multiple curves of variable length for multiple runs of the same algorithm on the same

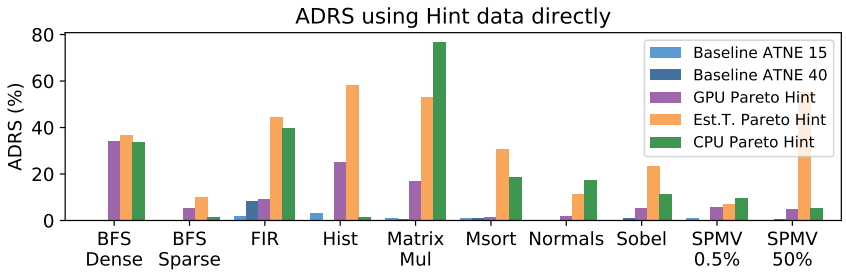


Figure 4.7. Prediction errors (ADRS) for different approaches: using ATNE without hint data, directly using Pareto hint data with GPU performance / pre-PnR estimations / CPU performance.

benchmark. Additionally, not all benchmarks have the same size, which is why the x axis presents the number of samples as a percentage of the total design space. This percentage also varies across benchmarks since we have a fixed number of initial samples, and always sample one design at a time. In order to summarize the results and present an average of this metric, we have several options: we could either extrapolate missing values to match the longest curve (i.e., use the last ADRS value for all remaining x axis values), or simply ignore missing values. We choose the latter option, as we have not noticed any significant difference between the two possibilities. However, this issue means that values at both curve extremities can be slightly less accurate.

4.7 Results

4.7.1 Directly using hint data

Here we present the results of simply sampling designs based on the hint design space as mentioned in Section 4.5.1. First, we calculate the Pareto optimal designs on the hint space, and compare the ADRS obtained by using this method to the ADRS obtained by running the baseline ATNE. We can see in Figure 4.7 that only using the hint Pareto designs leads to a large error. However, this methods generally selects few designs, while ATNE selects a much larger number. Therefore, we also compare the same method, but using various margin values when calculating the Pareto front. As mentioned before, using a margin is not always reliable, as it is difficult to

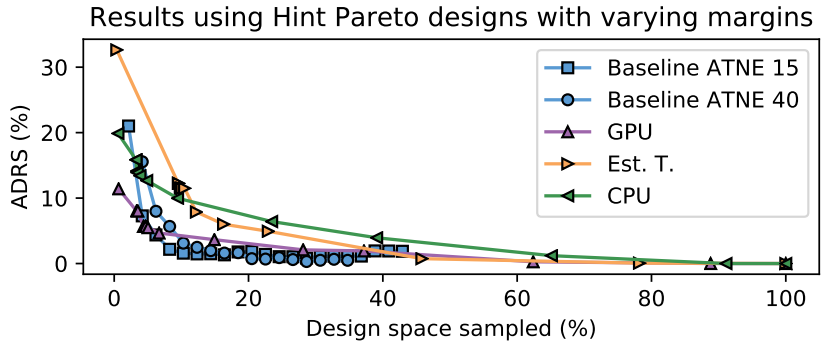


Figure 4.8. Comparison of the sampling/ADRS tradeoff between the baseline algorithm and the method selecting Pareto designs from hint data with various margins. All the results are averaged across all benchmarks.

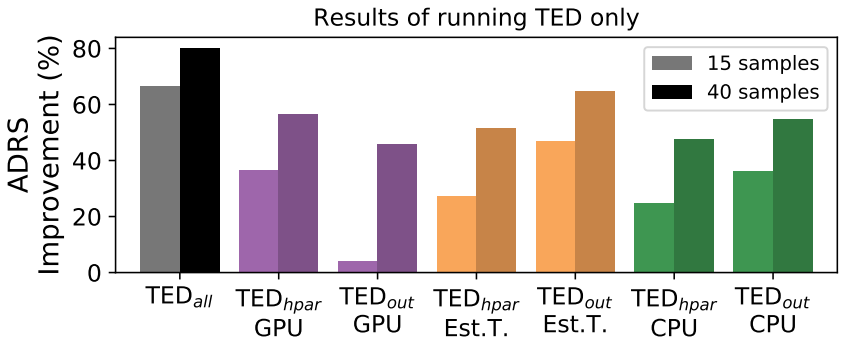


Figure 4.9. Improvement of modified versions of TED over the original TED in terms of ADRS for 15 and 40 samples. The results are averaged over all the benchmarks. **TED$_{\text{hpar}}$** uses normal TED plus the Pareto hint data, **TED$_{\text{out}}$** uses TED on the output hint space, and **TED$_{\text{all}}$** uses TED on all the output hint spaces.

control the number of sampled designs, but we are still interested in the performance of such a method in average over all the benchmarks. We selected multiple margin values between 0% (strict Pareto front) and 100% (all designs selected), and averaged the resulting number of selected samples and ADRS across all benchmarks. We compare this curve to the baseline ATNE curve in Figure 4.8. Most of the time, the tradeoff between number of samples and ADRS is worse when using this technique, although the values on the left of the curve indicate that the hint data can help with initialization.

4.7.2 TED Only

In these experiments, we modify TED as described in Section 4.5.2. **TED$_{\text{hpar}}$** is TED + Pareto Hint, **TED$_{\text{out}}$** is TED running on the output hint space, and **TED$_{\text{all}}$** is TED running on all the output hint spaces. Here we run the TED algorithm only, without the DSE framework. TED takes as input the number of designs to select. We choose the number of initial samples that we selected for the DSE framework, as we are aiming to improve the initialization of the DSE. Figure 4.9 reports the improvement of the various modified TED (target) over the original TED (baseline), in terms of ADRS, and averaged over all the benchmarks. In average, the improvement is always positive, which means that the hint data helps TED selecting designs that are closer to the Pareto front than the original TED for the same number of designs. The improvement is also much higher when we select a higher number of samples. We have established that hint data can improve TED, and now we want to apply these experiments to the machine learning ATNE framework, and combine them with our hint elimination method.

4.7.3 ATNE

The results of modifying TED are generally positive, but the goal TED is not to find the Pareto-optimal designs. We want to use these modified TED algorithms to give the entire machine learning DSE a better start. We run more experiments on the ATNE framework, by using our modified TED as initializer, and by applying the hint elimination algorithm described in Section 4.5.3. We measure the improvement in terms of ADRS and sampling complexity after convergence, and we also calculate the combined metric "difficulty". We experiment using the hyperparameters described earlier, and we present the results for $\alpha = 0.999$, 15 initial samples, and 40 initial samples. Figure 4.10 (a) shows the results for $\alpha = 0.999$ and 15 initial samples, averaged over all benchmarks. Overall, the results are very positive with few exceptions. In general, using modified TED only does not lead to a high improvement. While the error metric is vastly improved in some cases, it is often at the expense of sampling complexity, which translates

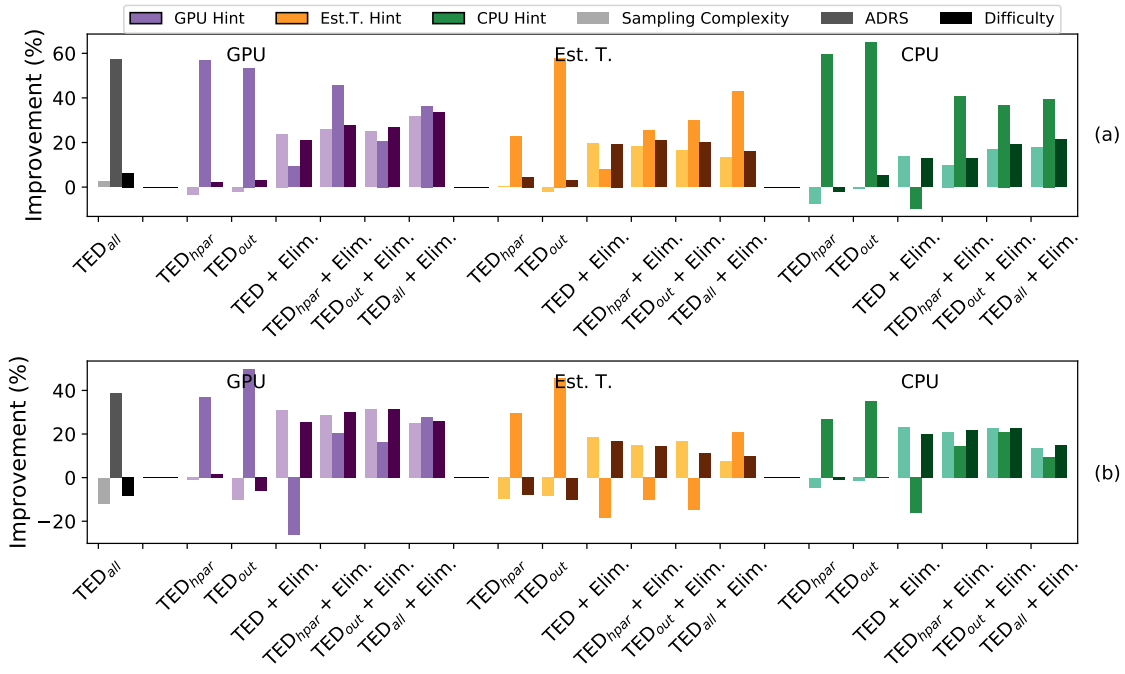


Figure 4.10. Improvement of various methods over the baseline ATNE algorithm in terms of ADRS, sampling complexity, and the combined *difficulty* metric. Here we use $\alpha = 0.999$. (a) uses 40 initial samples and (b) uses 15 initial samples. The results are averaged over all the benchmarks. We test the three modified version of TED and the original, combined or not with the hint elimination algorithm (*Elim.*). Each method is tested with the hint design spaces from GPU, Estimation Throughput, and CPU.

into only a small difficulty improvement or none at all. However, once added the hint elimination algorithm, the overall improvement is much more obvious.

The results when using only 15 initial samples are presented in Figure 4.10 (b). This graphs shows more negative results when using modified versions of TED only, which is a logical consequence of using less initial samples, and therefore modified TED showing less improvement (see Figure 4.9). The main reason for the negative difficulty improvement is that modified TED algorithms might sample more toward the Pareto front, but leaving large portions of the design space unknown, which turns into a low-confidence area that must be sampled by the ATNE algorithm, hence the large increase in sampling complexity. This low-confidence area can be pruned as a non-optimal space by the hint elimination algorithm, which leads to much better results. However, as shown in Figure 4.10, if the hint elimination algorithm is used too

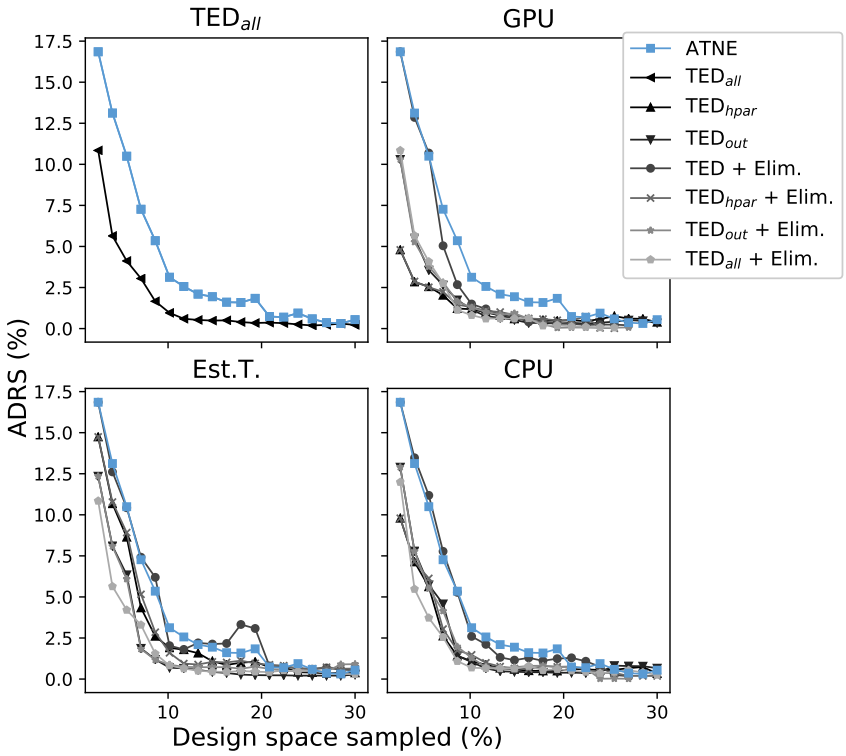


Figure 4.11. Curves showing the change in ADRS in function of the number of sampled designs, in average over all benchmarks. Each curve is a different algorithm, with $\alpha = 0.999$ and 40 initial samples. The curves are grouped by hint data type, with the addition of TED$_{all}$ which uses all hint types. Note that because each curve is an average, and not all runs of the algorithm converge to the same value, the curve can sometimes go up.

early without the help of hint-based modified TED (TED + Elim.), the error tends to increase. Indeed, the elimination step can sometimes underestimate the optimality of certain designs when the difference between these ground truth designs and their corresponding hint designs is too large, and they are situated in an under-sampled portion of the space. This flaw is solved when using the modified TED to obtain better information on these areas. These results underline the necessity of designing complex methods to extract information to avoid hurting the performance by feeding misleading data to the algorithm.

Fig. 4.11 and 4.12 show the change in ADRS as the different versions of the algorithm sample more designs. These curves cannot be compared directly with the plots in Figure 4.10, because the plots use the ADRS after the algorithm has finished sampling, while these curves

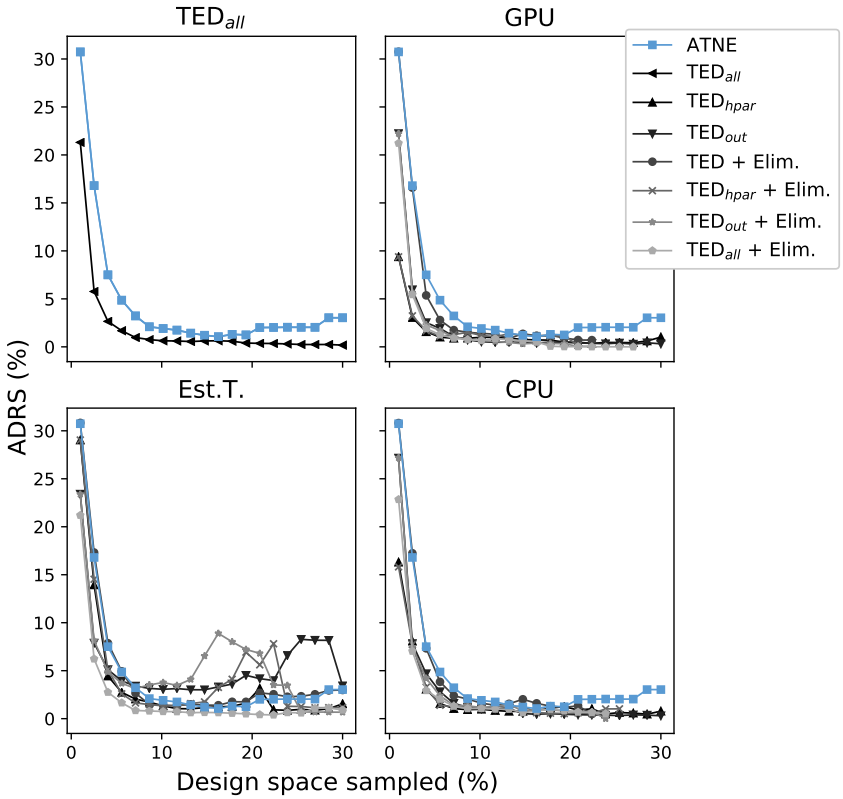


Figure 4.12. Curves showing the change in ADRS in function of the number of sampled designs, in average over all benchmarks, with $\alpha = 0.999$ and 15 initial samples.

present the average ADRS for different sampling ratios. Moreover, as explained in Section 4.6.5, the curves are not perfectly accurate due to the average and convergence differences, but we can still observe general trends. The hint elimination algorithm by itself does not improve the ADRS/sample curve by any significant amount, as it is designed to help the algorithm converge faster, rather than improve the sampling pattern. However, once combined with a modified TED initialization, it tends to decrease the error faster. The algorithm using the estimation throughput hint data with 15 initial samples show some limitations of our methods. With the Est.T. hint space, some techniques have a higher ADRS in average after sampling more than 7-8% of the design space. This is due to the modified TED versions picking bad starting points, as the Est.T. hint space is less accurate than the other two. This observation is supported by the TED$_{all}$+Elim curve which performs better as it can use all the information from the other hint spaces.

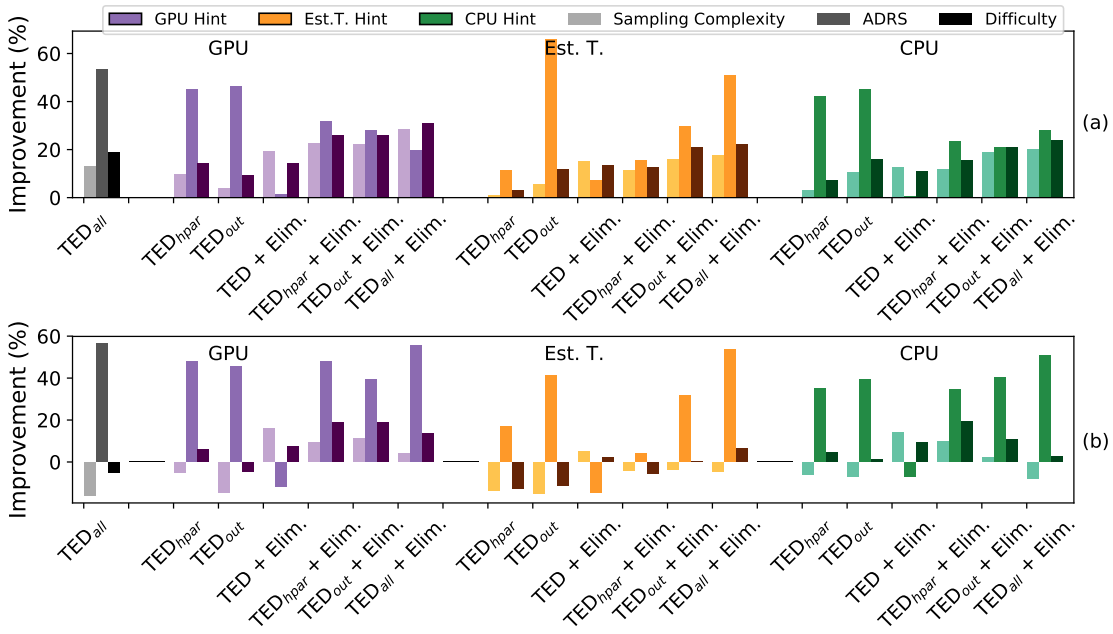


Figure 4.13. Improvement of hint data methods over the baseline ATNE algorithm for $\alpha = 0.2$, with (a) 40 initial samples and (b) 15 initial samples.

4.7.4 ATNE with low alpha

The results when using $\alpha = 0.2$ are very similar as previous results, and the conclusions remain the same. Choosing this alpha value means that the baseline ATNE algorithm is more "aggressive", i.e., has a lower threshold to eliminate designs and generally eliminates more designs at each iteration. Figure 4.13 shows the improvement of our methods over the baseline ATNE with this alpha. The improvement is similar as before. Only the results using 15 samples and estimation throughput (Est. T.) hint data are slightly worse, due to less accurate elimination suggestions based on this hint space, combined with a less stable algorithm. With a low alpha, the regression model tends to be based on less ground truth points, hence the need for a better hint to improve the accuracy of the elimination process.

4.8 Conclusion

We have shown that externally acquired *hint* data can improve FPGA design space exploration in average. It can reduce the sampling complexity to save hundreds of hours of

computation, and it can increase the accuracy to find more optimal designs. Although hint data can come from any source (CPU, GPU, various models, etc.), we also show that the useful information need to be extracted with care. Simple methods may not always work, and sometimes hurt performance, but smart data mining will improve the results. While our algorithms can be made more robust to uncorrelated hint design spaces, they all show promising improvement. These techniques can be generalized to other frameworks to further improve the value of information that can be collected from hint data.

Acknowledgements

Chapter 4, is coauthored with Althoff, Alric; Meng, Pingfan; and Kastner, Ryan. The dissertation author was the primary author of this chapter.

Chapter 5

Sherlock: A Multi-Objective Design Space Exploration Framework

5.1 Introduction

The process of optimizing a hardware design is usually a lengthy tuning of parameters that control various aspects of the design. One can tune the amount of parallelism, pipelining, memory caching, etc. to balance the resulting throughput, area utilization, and power consumption. Every time a modification is made, if the designer wishes to evaluate their design to obtain precise measurements, they need to compile and run it. This process can easily take multiple hours. A complex design can comprise of many parameters, rendering the design space exploration procedure very slow and tedious.

Certain software applications present the same issue when it comes to parameter tuning. Many machine learning or computer vision applications contains tens or hundreds of parameters, along with large datasets which makes any evaluation very slow. In either hardware or software design, all these parameters create a highly complex design space that is often non-linear [42, 85]. Such a design space generally contains mutually exclusive objectives (e.g., accuracy vs. throughput). It is therefore interesting for the designer to find the set of optimal designs along all the objectives. In a context where compiling and/or running each design takes a long time, evaluating all possible designs is too time-consuming and sometimes impossible.

We develop a machine learning framework called Sherlock, that utilizes active learning

to intelligently select designs to evaluate. Sherlock focuses the learning on the set of optimal designs. Understanding the shape of the entire space can be useful when analyzing the effects of different parameters on the design, however, in the case of design optimization, we want to obtain the optimal set as fast as possible. Sherlock can reach the optimal set quickly by minimizing the initialization size, and performing a sample selection entirely driven by the estimated optimal set, using a strategy that balances exploration and exploitation based on expected results.

The active learning process is based on a regression model that iteratively provides estimates of the entire space. We test Sherlock using multiple types of regression models, spanning from complex models often used in active learning literature, to simpler consensus-based interpolation kernels that help reaching an almost optimal solution faster. In order to make our framework more flexible, we also design a model selection strategy based on the Multi-Armed Bandit problem, that rewards the models directly improving the actual Pareto front. With this strategy, the framework can quickly decrease the importance of models that do not provide correct estimates. It can then leverage all models relative to their positive contribution to reach the optimal designs faster.

The contributions of this work are:

- We develop a design space exploration (DSE) framework that targets the optimization of FPGA designs, but that also generalizes to other applications with slow evaluation time.
- We outfit our DSE framework with a model selection strategy to adapt it to a wide variety of design spaces.
- Against similar state-of-the-art frameworks, we improve the convergence toward the Pareto front on several FPGA applications and three complex software application.

We discuss related work in Section 5.2. In Section 5.3, we introduce the problem and notations, and we describe the core active learning algorithm. In Section 5.4, we present our

method to select regression models. Section 5.5 contains the results of running Sherlock on multiple datasets, and we conclude in Section 5.6.

5.2 Related Work

The Design Space Exploration topic in literature is often focused on hardware design, a very time-consuming and costly process. FPGA design is a popular topic, especially in recent years with the improvements of High-Level Synthesis (HLS) tools that render the design parameterization easier, and open the field to designers with less expertise in hardware design. In these cases, DSE techniques allow one user to specify potential optimizations, and determine the best specifications to produce the most optimal architectures. The automation aspect of DSE is what drives algorithms to attempt to reach the optimal solutions faster.

Several strategies exist to explore design spaces without sampling all the designs. Evolutionary design space exploration [93, 26, 81] uses genetic algorithms to converge toward an optimal solution. Particle swarm optimization [91, 80] generates a population of particles searching the space of designs and using various metrics to advance toward the optimal solutions. These techniques require a large number of samples to converge, and therefore are more adapted to problems where the evaluation of each sample is a fast process, but the space is too large to be evaluated entirely, which usually not applies to hardware optimization.

In hardware design, architectures often take several hours to compile on a multicore machine. In these scenarios, the designer does not wish to explore the entire space. Specifically, the number of samples is very important, as each single design not sampled can save hours of optimization. To solve this problem, one solution is to substitute the design space for an approximation. Predictive-based methods for DSE replace the actual design by an analytical model. By using such a model, a rapid exploration of the space can be performed, either by exploring the entire space, or using an evolutionary approach. [75] designs a scheduling system to predict and efficiently simulate the designs, to quickly explore the design space of

the analytical model. [117, 115] analyze the structure of FPGA kernels generated by High-Level Synthesis (HLS) using the OpenCL language, and build a compute and memory model that can be use as a surrogate to significantly speed up the DSE process. Their performance prediction error varies between 4 and 16%. [99] explores the design space at the pragma level by building a probabilistic model for each type of pragma. They perform a fast exploration by using an ant colony optimization algorithm, and obtain an error of 1.7% on SystemC benchmarks. [127] and [126] propose tools to analyze the structure of the HLS code directly before the synthesizing steps occurs. Their model achieve a 400×-4000× speedup in DSE, and a final DSE accuracy of 95%. Generally, predictive-based methods exploit features specific to the optimized application. These specific features make predictive approaches difficult to generalize to other types of problems. In our work, we aim to build a tool that applies to a wide variety of problems, and therefore we do not choose the predictive approach.

Often opposed to predictive-based methods, evaluation-based methods measure the exact quantity of the target objectives to optimize. In hardware design, this method consists of compiling an architecture specification to the actual hardware, then running the compiled design onto an application-specific dataset, and measure the throughput, area utilization, power consumption, and other potential optimization goals. These measurements give a very accurate representation of how the design actually performs and how it compares to different designs at the cost of long processing time.

One solution to accelerate this evaluation process is to parallelize it, and employ a smart division of the space to attribute the computing resources. [119] proposes a Multi-Armed Bandit (MAB) algorithm to balance the computing resources between different portions of the space, and leverages the exploration-exploitation tradeoff of MAB to iteratively allocate more resources to the optimal subspaces.

Another popular solution is to use iterative machine learning algorithms, such as active learning [74, 132], to minimize the number of designs to evaluate. [77] proposes an active learning framework called ATNE based on non-Pareto elimination. ATNE creates multiple

regression models from different subsets of the known data, then computes an elimination threshold from the variance of the predictions. Based on this threshold, designs predicted to be dominated are eliminated from consideration. The algorithm then samples a new design and reiterate until convergence. This technique allows a much faster convergence toward the Pareto front than previous techniques. The Hypermapper framework described in [11, 85] performs active learning by modeling known designs with a random forest regression algorithm, and simultaneously sampling all the predicted Pareto-optimal solutions. The algorithm iterates until a sampling budget is reached. This framework is more optimized toward very large design spaces where it is still reasonable to perform a high number of design evaluations (100 to 300 samples per iterations). Their framework is applied to the optimization of Simultaneous Localization And Mapping (SLAM) algorithms. The ε-Pareto Active Learning (ε-PAL) [131] is an improvement over PAL [132], and uses Gaussian Process as a regression model in order to predict an uncertainty region for each predicted design output. By using these uncertainty regions, the algorithm can discard non-optimal designs with high accuracy, and progressively build a predicted Pareto-optimal set with an ε margin. The sampling process is based on minimizing the uncertainty of the predictions.

Our work is similar to ε-PAL and ATNE, but without the pruning step, which can potentially eliminate optimal designs from consideration, especially when the regression model is not adapted to the design space being searched. Another similar work is Flash [84], a method to explore the space of possible configurations for software systems. Flash uses decision trees to iteratively sample configurations. While our work focuses primarily on optimizing hardware design, we show that it can work on software design spaces as well, and perform better than Flash in most cases.

Additionally, these algorithms rely on well-known regression models, such as Random Forest or Gaussian Process, that may not perform well on all types of design spaces. Our framework uses simpler regression kernels by default, and we propose a solution to automatically utilize more complex models for design spaces that require it.

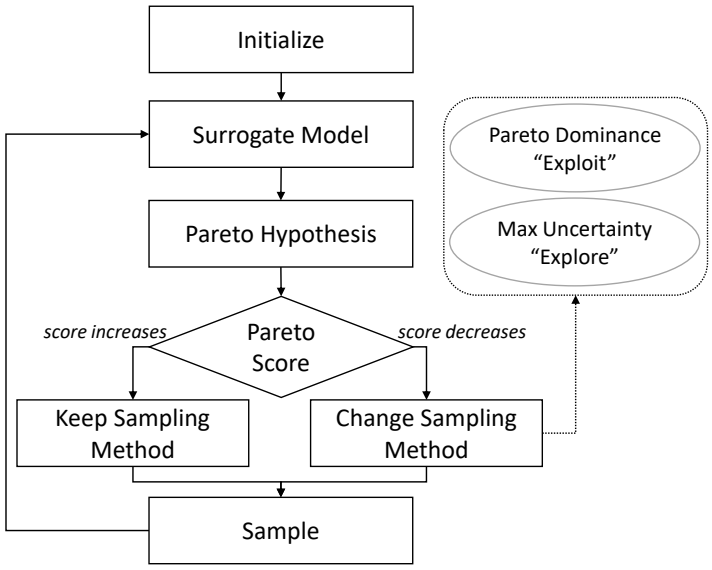


Figure 5.1. Sherlock workflow.

5.3 The Sherlock Algorithm

Sherlock implements an evaluation-based strategy for Design Space Exploration (DSE) that utilizes an active learning technique to iteratively improve the known set of optimal designs. First we define the problem and introduce formal definitions, then we present the active learning algorithm and how its different components interact.

5.3.1 Scope and Definitions

Sherlock is a DSE framework that can target any application. As such, it only considers an abstract representation of that application, through its *design space*. A design space of an application is composed of both an *input space* and an *output space*. The input space is the set of all possible variations of the application that are functionally identical. These variations are created through the use of parameters, also known in the DSE literature as *knobs*. Knobs take n values, for $n \in [2, \infty[$. They can be discrete, categorical, or continuous. Our framework requires a finite pool of candidates. Continuous knobs can generally be discretized by knowing the bounds of the knob and choosing a reasonable set of values based on the target platform (e.g., powers of

two, regular grid, etc.). Categorical knobs are interpreted as numerical values. The input space X is then defined as $X = \{k_1 \times k_2 \times ... \times k_n\} \in \mathbb{X}^{m \times n}$ where k_i is a knob vector containing all the possible values for this knob, and in this case, $\mathbb{X} = \mathbb{R}$. The resulting matrix has n columns for each knob, and m rows for each unique and valid combination of knob values; in other words, m design candidates.

The output space is defined by the optimization objectives set by the designer of the application, such as throughput, accuracy, power consumption, etc. The output space $y \in \mathbb{Y}^{m \times o}$ is a matrix of m rows for each design candidate, and o columns for each optimization objective. The final design space S is the combination of the input and output space: $S = \{(X_i, y_i)\}$.

The problem is defined as follow: X is known by the designer, but y is unknown. What is the set $P \subseteq S$ of design candidates that are optimal on all objectives? As a multi-objective optimization problem, this set P corresponds to the set of Pareto optimal design, i.e., designs that can not be improved on one objective without decreasing another, also known as the Pareto front.

5.3.2 Active Learning

In order to find the Pareto optimal designs, Sherlock uses a well-known machine learning technique known as active learning. Active learning works iteratively by 1) creating a surrogate model to formulate an hypothesis of optimal designs, 2) selecting a candidate to sample, 3) sampling the chosen design to obtain objective values, and 4) refining the model based on the new sample. The loop continues until a stopping criterion is met. The specifics of Sherlock's active learning workflow are summarized in Fig. 5.1 and described below. The pseudo-code is also presented in Algorithm 5.

Initialization

As a starting point of the algorithm, we choose an initial number of samples to evaluate and collect their objective values. We build a set K of known designs (for which y_i is known)

that will grow with future iterations of the framework.

$$K = \{(X_1, y_1), (X_2, y_2), ...\} \tag{5.1}$$

Before the initial sampling, we do not possess any information on the output space, therefore the sample selection must occur based on the input space only. This can be achieved by random selection, but we choose the Transductive Experimental Design (TED) [121] algorithm as it attempts to provide a representative set of experiments.

Formulation of Pareto Hypothesis

The goal of Sherlock is to focus the learning method on the Pareto front of the design space. In this step, we try to obtain an estimated Pareto front through the use of a surrogate model. A surrogate model is defined as a function $\hat{f}$:

$$\hat{f} \leftarrow g(K) \tag{5.2}$$

$$g : S \rightarrow (\mathbb{X} \rightarrow \mathbb{Y}) \tag{5.3}$$

where g is a supervised learning method for regression. We use the surrogate model to obtain the estimated Pareto front, along with a measure of the uncertainty of the estimation:

$$\hat{y}, H \leftarrow \hat{f}(X) \tag{5.4}$$

$$\hat{P}, H_{\hat{P}} \leftarrow \text{pareto}(\hat{y}, H) \tag{5.5}$$

where $\hat{y}$ is the estimation of the output space, H is the uncertainty of each estimation, and *pareto* is a function to extract the set of Pareto optimal designs.

Sample Selection

Sampling is the process of choosing one design s_i to evaluate and obtain its output value. The result is an increased set of known designs:

$$K \leftarrow K \cup \{s_i\} \tag{5.6}$$

We need to determine the index i of the design to sample. Sherlock focuses on sampling designs on the Pareto front. In order to reach these designs, the algorithm has two options: increase the understanding of the space near or on the Pareto front (*explore*), or sampling directly the estimated Pareto front (*exploit*). The *explore* sampling mode chooses a design that has been estimated by the surrogate model with a high uncertainty:

$$i \leftarrow \text{argmax}(H_{\hat{P}}) \tag{5.7}$$

This mode increases the confidence of the estimated Pareto front. The *exploit* mode selects a design with a high estimated Pareto dominance:

$$i \leftarrow \underset{s_i \in S \setminus K}{\text{argmax}}(\text{scores}(\hat{y})) \tag{5.8}$$

If the model is a good estimation of the space around the Pareto front, this step leads to picking a more optimal design.

We choose between the two sampling modes based on the improvement of the predicted Pareto front. We attribute a score to the prediction from the model, and monitor its changes. The score function is presented in Algorithm 6 and consists of comparing the output value of each design (scaled by the number of designs) against the sum of the output of all other designs. We apply this function to the estimated Pareto front. We compare the current maximum score to the maximum score from the previous iteration. If the score is decreasing, we switch the sampling

Algorithm 5: Sherlock Algorithm

Input : X

$s_1, s_2 = TED(X)$
$K = \{s_1, s_2\}$
mode $\leftarrow$ "explore"
prevscore = $-\infty$
while $|K| <$ *sample budget* **do**
  $\hat{f} \leftarrow g(.;K)$
  $\hat{y}, H \leftarrow \hat{f}(X)$
  $\hat{P}, H_{\hat{P}} \leftarrow$ pareto$(\hat{y}, H)$
  curscore $\leftarrow \max_{s \in \hat{P}}$(scores$(\hat{y})$)
  if *curscore ¡ prevscore* **then**
    mode $\leftarrow$ next(mode)
  end
  if *mode = "explore"* **then**
    $i \leftarrow$ argmax$(H_{\hat{P}})$
  else if *mode = "exploit"* **then**
    $i \leftarrow$ argmax$_{s_i \in S \setminus K}$(scores$(\hat{y})$)
  end
  $K \leftarrow K \cup \{s_i\}$
  prevscore $\leftarrow$ curscore
end

Algorithm 6: Score Algorithm

Input : y
Output : Scores: array of size m

for $j \in [1..o]$ **do**
  Sum$[j] = \sum_i y[i, j]$
end
for $i \in [1..m]$ **do**
  Scores$[i] = \sum_j (y[i, j] * m -$ Sum$[j])$
end

mode.

5.3.3 Surrogate Model

The surrogate model must be a supervised regression algorithm that can provide a prediction uncertainty, which is used to refine the global prediction in subsequent iterations. There exist two major methods to provide an uncertainty in a regression model: 1) with ensemble

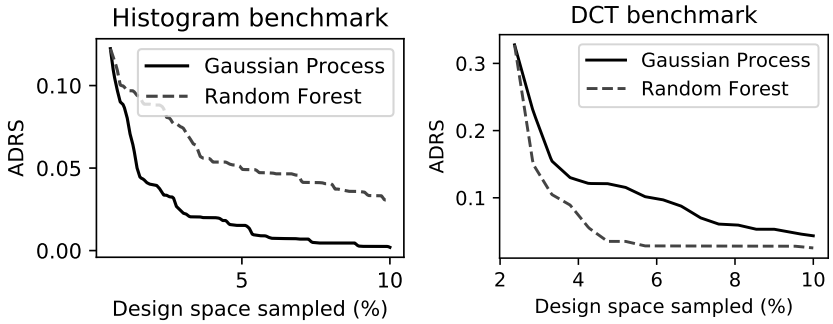


Figure 5.2. Illustration of the difference of performance using Sherlock with two regression models on two benchmarks.

technique, and 2) with Bayesian learning.

Ensemble techniques create multiple regression models, each with a subset of the known points. The models then output different predictions, which are aggregated by voting or averaging. This method can provide an estimation of the uncertainty of the prediction by computing the variance over all the models. A popular ensemble model is the Random Forest predictor based on a set of decision trees, and is used in several active learning frameworks.

Bayesian learning techniques leverage the Bayes theorem to progressively update statistical distributions based on provided evidence. Typically, a regression model starts with a prior distribution over its weights, and combines it with the likelihood from known points to create a posterior distribution. The parameters of the posterior distribution can be used to compute a measure of uncertainty. A popular example is Gaussian Process models that create a prior distribution over functions by using a kernel to express the correlation between points.

Sherlock can use any of these types of learning algorithms. We experiment with Random Forest and Gaussian Process as they can generally model complex design spaces, and we also create a consensus-based Radial Basis Function interpolator, that provides kernel-based interpolation of unknown data, and is faster to compute than a Gaussian Process.

5.4 Model Selection

Most design spaces are inherently different as they represent a wide variety of relationships between input variable and output design goals. Some design spaces can be modeled by a simple linear equation, while others require a more complex model. This is reflected by the performance of different surrogate models in active learning frameworks. In Figure 5.2, we present the results of running Sherlock on two different benchmarks, by plotting our error metric (ADRS) against the percentage of design space sampled. We give more details on the experimental setup and the results in Section 5.5. Here we want to illustrate the difference between the use of two different models. In the first design space, using a Gaussian Process causes the algorithm to converge toward the Pareto front faster, and in the second example, the Random Forest makes the algorithm converge faster.

A common solution to pick a model is to test the algorithm on similar design spaces, and choose the most efficient one, possibly by cross-validation. Instead, we propose to learn the best model using a Multi-Armed Bandit strategy that iteratively updates the importance of each model based on the Pareto set improvement.

5.4.1 Algorithm

At each iteration of the active learning process, we want to choose a surrogate model g among a small pool of models $G = \{g_1, g_2, ...\}$. We start with no prior knowledge of which model performs better, and we want the framework to iteratively increase the importance of models that generate good results.

A solution to this problem is to use one of the well-known Multi-Armed Bandit (MAB) strategies. The MAB problem is a Bayesian optimization problem where we wish to determine the distribution of independent variables with unknown outcome (the bandits), and choose the variable providing the best outcome. The various MAB algorithms provide a tradeoff between exploitation (observing the bandit with best known outcome), and exploration (observing other

Algorithm 7: Model Selection Algorithm

Input : Models $\{g_1, g_2, ..., g_i, ...\}$, Reshape factor r

Initialize: $\alpha_i = 1, \beta_i = 1 \; \forall i$
while $|K| <$ *sample budget* **do**
  $P_i(\theta) = Beta(\alpha_i, \beta_i) \; \forall i$
  $\hat{\theta}_i \sim Beta(\alpha_i, \beta_i) \; \forall i$
  $i = \text{argmax}(\hat{\theta}_i)$
  Choose model $g = g_i$
  `// Sherlock algorithm`
  Compute $Hv = \text{hypervolume}(K)$
  $x = Hv > \text{prev}(Hv)$
  $\alpha_i = \alpha_i + x * r$
  $\beta_i = \beta_i + (1 - x) * r$
end

bandits to refine their distribution).

In this case, we consider each model as a bandit. The outcome of observing one bandit is either an improvement in the current Pareto set, or no improvement. In other word, we are trying to learn a Bernoulli distribution for each model. Consequently, we can select the prior distribution of the bandits as a Beta distribution. We define the prior distribution with parameter θ for each model i as $P_i(\theta) = Beta(\alpha_i, \beta_i)$. We update these distributions by selecting one bandit and observing the outcome. A good choice of sampling algorithm is Thomson Sampling [108] that provides a good tradeoff between exploration and exploitation [22]. The algorithm draws a random sample from each distribution: $\hat{\theta}_i \sim Beta(\alpha_i, \beta_i) \; \forall i$, then chooses the bandit with the highest sample. The observation x of the selected bandit corresponds to the improvement of hypervolume over the known designs (hypervolume(K)), after we sample a design according to a strategy as defined in Section 5.3.2. In other words, if the model g_i improved the Pareto set, x is a positive outcome, i.e., $x = 1$, otherwise $x = 0$. We compute the posterior distribution based on the outcome, and use it as prior for the next iteration.

Algorithm 7 shows the details of the method and how it integrates with the Sherlock algorithm described in Algorithm 5. Note that we use an optional posterior reshaping factor r that changes the variance of the distributions. As a result, increasing the value of r favors exploitation

over exploration (i.e., the model providing the best outcome gets selected more often), and the policy becomes more greedy. Increasing this value also has the side benefit that each positive outcome is given more consideration, and potential improvements from models later in the sampling process will re-adjust their importance faster. It provides a small chance to switch the most important model during the sampling process.

5.5 Results

5.5.1 Experimental Setup

We implement Sherlock using Python 3 with the numpy and scipy libraries. We implement four types of surrogate models: a Gaussian Process (GP) with a Matern kernel using the GPy library, a Random Forest (RF) from the scikit-learn library, and Radial Basis Function (RBF) interpolation algorithms with consensus decision similar to the implementation in scikit-learn, with both a multiquadric basis and a thin plate basis. We compute an error metric based on the ground truth design spaces, using the Average Distance to Reference Set (ADRS) metric [92]. ADRS measures the average normalized distance between the estimated Pareto front and the reference Pareto front. It is a normalized value, sometimes reported as a percentage. The closer it is to 0, the better the estimation is. We are particularly interested in the evolution of the ADRS value as we sample the design space. We measure our results by plotting the ADRS curve in function of the number of samples. For a better comparison between benchmarks, we normalize the number of samples to the size of the design space, and report the results in function of the percentage of the space sampled. The goal of Sherlock is to produce a curve that converges to zero as fast as possible. To summarize our results, we compute the area under the curve for a section of the curve (up to a certain percentage of the space, e.g., 20%), as a measure of how fast the algorithm converges toward a good solution.

We compare our results to the ATNE algorithm [77] implemented in Python 3, the ε-PAL [131] algorithm implemented in MATLAB, and the Flash [84] algorithm implemented in

Python 2. For all algorithms, we set the number of initial samples to five. In ATNE, we monitor the ADRS for each design sampled until it converges and stops. For ε-PAL, we set the ε to zero, monitor the ADRS curve for each sample, and complete the curve with the samples from the estimated Pareto set after the algorithm converges. All frameworks are run multiple times on each benchmark, and we compute the average ADRS curve.

5.5.2 Dataset

We test our algorithm on a set of FPGA benchmarks that cover different types of applications. We get our dataset from the Spector benchmarks [42]. These benchmarks are FPGA applications outfitted with knobs that can be tuned, along with a test dataset to run them and measure their throughput. The knobs are software-defined parameters that translate into architecture changes. They cover typical FPGA optimizations such as pipelining, unrolling, partitioning, parallelizing, and also some optimizations specific to each application (sliding window width, etc.). The different combinations of values for knobs create unique designs that are functionally equivalent, but produce a different outcome in terms of logic utilization and throughput.

Each design takes multiple hours to compile, making DSE an essential process to tune these applications. Each unique design of these application has been compiled and run to create exhaustive design spaces containing between 200 and 1500 designs each. We also add the Iterative Closest Point (ICP) design space from [44] that implements a Simultaneous Localization And Mapping (SLAM) algorithm on FPGA using OpenCL, and uses knobs to create a similar design space as in Spector. It contains 1276 designs. We use these provided design spaces that consist of the knob values, the actual FPGA area utilization, and the measured throughput of each design.

The goal of a DSE framework such as Sherlock is to search a predefined space for optimal designs. Therefore our ground truth consists of the set of Pareto-optimal designs in each exhaustive design space. We run Sherlock by considering that the outcome of each design is unknown and let the tool incrementally find and improve an estimated Pareto set. We can then

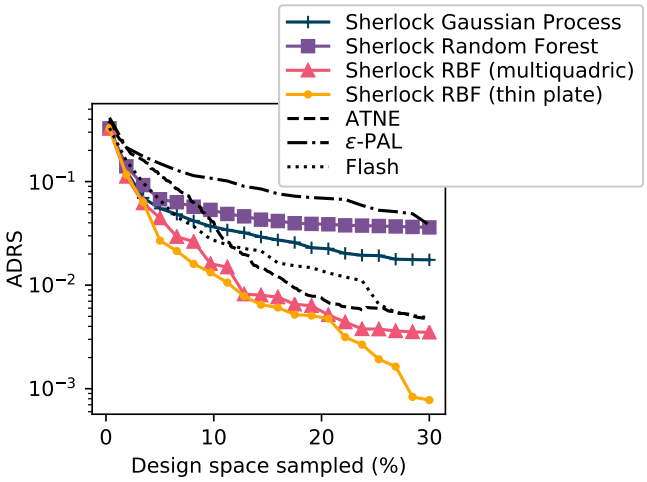


Figure 5.3. Average performance of several algorithms on all the FPGA benchmarks. We plot the error (ADRS - lower is better) against the percentage of design space sampled. We test Sherlock with multiple regression models.

compute the ADRS metric between the estimated Pareto set and the ground truth set that we defined initially.

5.5.3 Active Learning Results

We run the basic active learning algorithm in Sherlock over the FPGA benchmarks. We calculate the ADRS curve at each sample, based on the provided ground truth design spaces. Our results focus on the first 30% of the design spaces. 30% is a large budget for most applications with a slow evaluation time, and in our test cases, it is always sufficient to reach an ADRS below 1% with the best regression model. Moreover, we want to emphasize the convergence of the ADRS curve by measuring the area under the curve. This metric puts more weight at the beginning of the curve where we expect the error to decrease quickly, while fine-tuning optimizations still influence the metric, but to a lesser extent.

In Figure 5.3, we present an average curve of running Sherlock, ATNE, ε-PAL, and Flash on the FPGA benchmarks. We run Sherlock with different regression models. We can observe that all versions of Sherlock perform better or similar as other algorithms in the first 5% of the spaces, and then perform differently based on the chosen model. In average, using

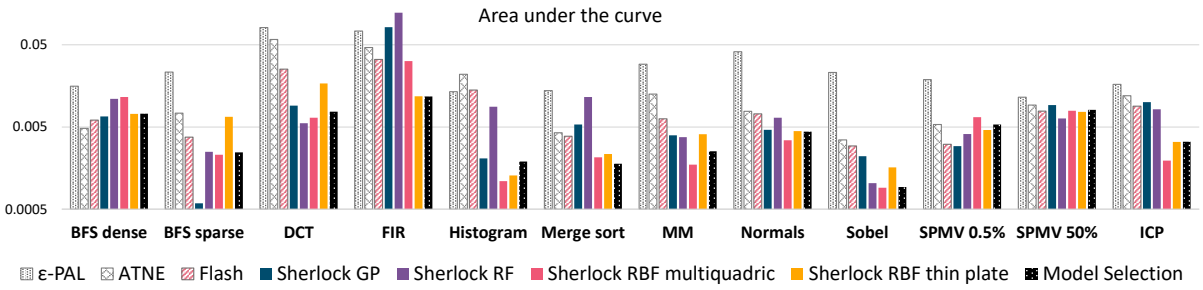


Figure 5.4. Performance of several algorithms on individual benchmarks. We compute the area under the ADRS curve as a measure of convergence, over the first 30% of design space sampled. A lower value means that the algorithm reaches a better solution faster. We compare Sherlock with different regression models to other state-of-the-art algorithms. We also compare with our proposed model selection algorithm.

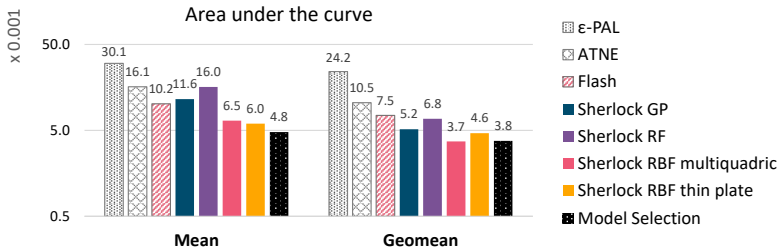


Figure 5.5. Average area under the curve for the benchmarks presented in Figure 5.4. We present the arithmetic mean, and the geometric mean to take into account the variability of the results. The values are scaled by 1000 for readability.

the RBF interpolator with a thin plate kernel tends to work better. However, the average curve does not reflect the performance on individual benchmarks, and is especially skewed by the FIR benchmark, which is more difficult for several versions of Sherlock.

We summarize the results for individual benchmarks by calculating the area under the curve (AUC) over the first 30% of the space. Figure 5.4 shows the AUC of Sherlock, ATNE, and ε-PAL for all benchmarks, and Figure 5.5 shows the average AUC over all benchmarks. A smaller AUC represents a faster convergence of the algorithm toward the true Pareto front. The best performing model in average (RBF with a thin plate kernel) produces an AUC 1.7$\times$ smaller than Flash, 2.7$\times$ smaller than ATNE, and 5$\times$ smaller than ε-PAL. However, the performance of each model varies with individual benchmarks. While we could reasonably pick the model with the best average result, we also want to study the case where we let the algorithm decide which

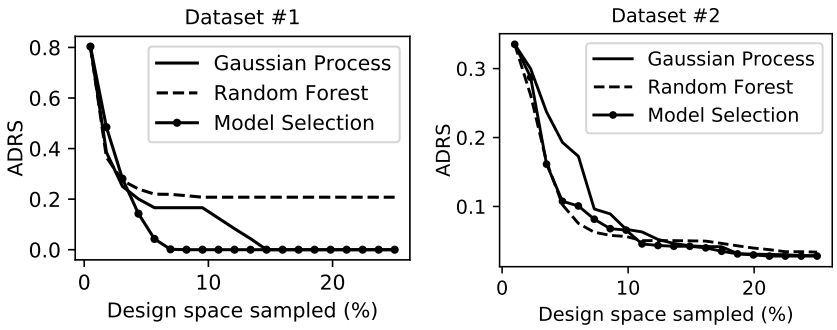


Figure 5.6. Performance of the model selection algorithm on two simulated datasets.

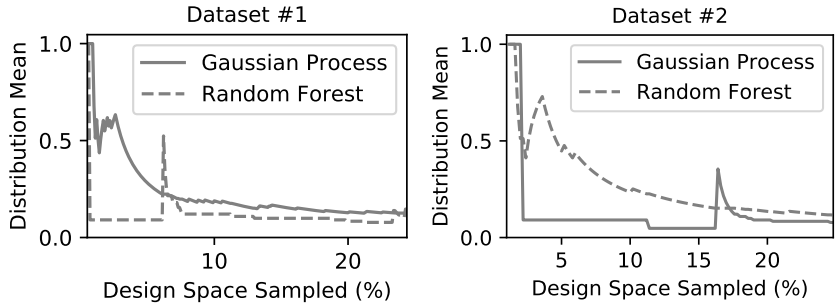


Figure 5.7. Calculated mean of the Beta distributions of the two models for the two simulated datasets.

model to choose.

5.5.4 Model Selection Results

Algorithm Analysis

We generate two synthetic design spaces, each optimized for a different regression model (Random Forest and Gaussian Process), and we run the model selection process on these datasets to verify that the algorithm can choose the proper model for each design space. Figure 5.6 compares the results of Sherlock using a single model and Sherlock using model selection. In both cases, the algorithm with model selection performs as good as the best model, or better. We also plot the calculated mean of the Beta distribution associated to each model when running model selection. Figure 5.7 shows the evolution of this mean. As expected, the model performing better keeps a larger mean. The spikes in the curve corresponds to the reshaping factor (set to 10) designed to amplify the increase of the mean when an model performs better only later in the

sampling process.

Sherlock Results

We run Sherlock with the model selection algorithm on the FPGA benchmarks. We present the AUC results in Figures 5.4 and 5.5, and the actual ADRS curves in Figure 5.8. In many cases, the performance of Sherlock with model selection is comparable to Sherlock with the RBF interpolator, and better in some cases. In average, this algorithm performs better than the other solutions, without having to choose a particular model. The AUC is about 2× smaller than Flash, 3× smaller than ATNE, and 6× smaller than ε-PAL. The Gaussian Process and Random Forest implementations of Sherlock do not converge very well in many benchmarks, as a result of being stuck in local minimum regions. The model selection process can more easily avoid these local minima by selecting a different model. Certain models such as FIR filter contains a local minimum that creates a high variance in the performance of different models, and the selection process clearly helps in this case. Conversely, a benchmark like SPMV 0.5% depends more on the learning rate of the chosen model, and the overhead of choosing between multiple models is more obvious.

5.5.5 Software Dataset

Certain software applications can be very slow to evaluate, while also containing a large number of knobs. Typically, computer vision algorithms can be complex and contain many possible configurations. We want to evaluate how well can our framework optimize Simultaneous Localization And Mapping (SLAM) and Visual Odometry (VO) algorithms.

We design a Visual Odometry (VO) algorithm using the OpenCV library. The main component in VO is the choice of visual features detector and descriptor. We choose three feature detectors/descriptors (SIFT, SURF, BRISK), and modify the parameters of each of them. We run the algorithm on a standard dataset with ground truth data, and measure the running time and the accuracy of the output (The accuracy is defined by the difference between the estimated

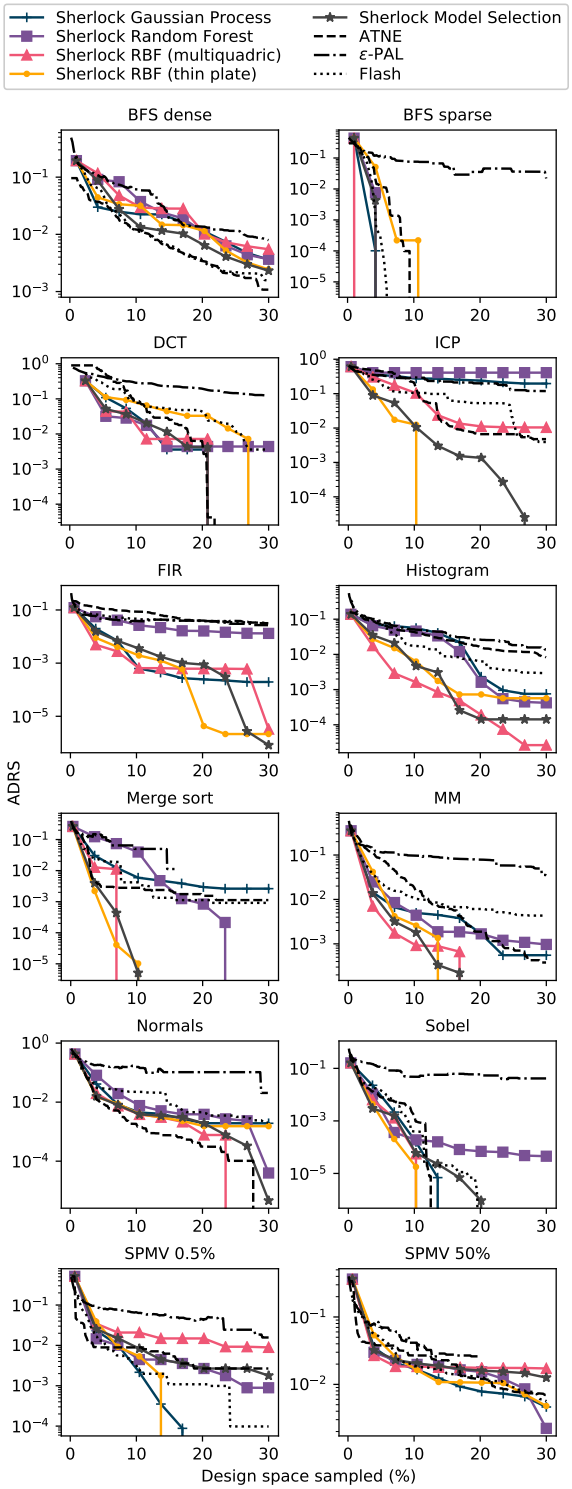


Figure 5.8. Comparison of the ADRS curves for multiple algorithms on all the FPGA benchmarks.

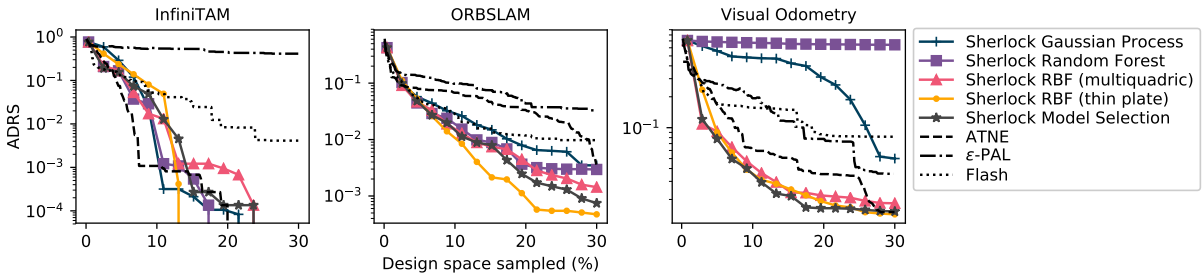


Figure 5.9. Comparison of the ADRS curves for multiple algorithms on the SLAM benchmarks.

trajectory and the ground truth). We create an exhaustive design space from an input space X using our defined knobs (choice of detector/descriptor algorithms and parameters for each of them), and an output space y using the measured throughput and accuracy.

Our SLAM design spaces are created using the SLAMBench 2.0 [12] framework. We run two different types of SLAM algorithms on a dataset with ground truth. We use InfiniTAM [58] which uses depth-based tracking (i.e., uses only depth information to track the motion of the camera), and ORBSLAM [82], which uses RGB images for tracking. Both algorithms have very different configurations to optimize. We create design spaces based on measure throughput and accuracy.

The ADRS curves are presented in Figure 5.9 and the area under the curve is summarized in Figure 5.10. Certain algorithms get stuck inside regions of design spaces and fail to explore the remaining space effectively, but Sherlock with RBF interpolators generally perform well, and the results from the model selection strategy follow this trend.

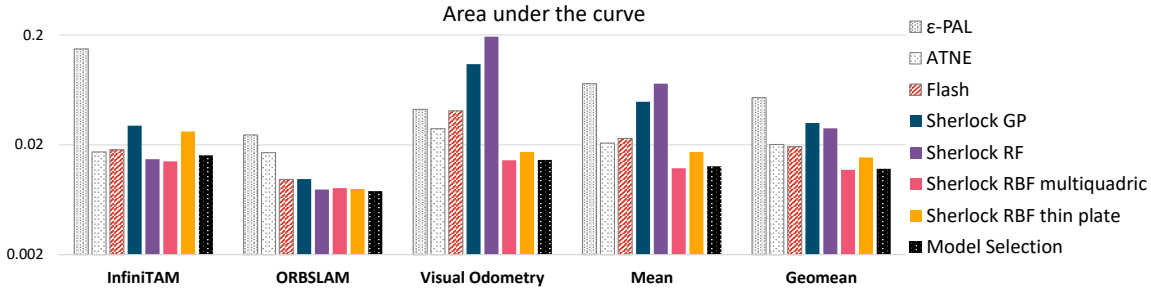


Figure 5.10. Area under the ADRS curve for the first 30% of the design space sampled.

5.6 Conclusion

In this chapter, we have presented Sherlock, our evaluation-based, multi-objective, Design Space Exploration framework, that improves over previous work on a wide variety of applications. Sherlock is an active learning algorithm, heavily focused on improving the set of optimal designs at each iteration, and as such converges very quickly toward a low-error solution. We have tested our framework with multiple regression models that present a wide variance in the quality of results on different benchmarks. In general, we have found that simple RBF interpolation function perform better than traditional Random Forest or Gaussian Process models on FPGA design spaces. We also proposed a Multi-Armed Bandit-based algorithm to render the framework agnostic to the type of regression model used, by iteratively selecting the most useful model on a per-application basis. The results of this model selection are consistent over multiple benchmarks, and provide a better average performance.

Acknowledgements

Chapter 5, in part, has been submitted for publication of the material as it may appear in the 28th ACM/SIGDA International Symposium on Field-Programmable Gate Arrays (FPGA), 2020. “Sherlock: A Multi-Objective Design Space Exploration Framework”. Gautier, Quentin; Althoff, Alric; and Kastner, Ryan. The dissertation author was the primary investigator and author of this paper.

Chapter 6

Conclusion

In this dissertation, we have addressed the problem of designing complex software and hardware applications through the design of a 3D scanning system for archaeological sites. First, we have manually explored several possible setups for the scanning system. It requires the combination of one or multiple sensors, with a SLAM algorithm to process the sensor data. We have quantified the accuracy of multiple hardware/software combinations after running a set of lengthy experiments that could only be applied to carefully chosen solutions. Then we have explored possible architectures to transfer a SLAM algorithm to a low-power FPGA platform. By analyzing thousands of FPGA designs, we have established various relationships between optimization parameters and optimization goals (throughput, logic utilization, etc.). However, the brute-force exploration of the design spaces we have created is generally infeasible to create an optimized architecture.

We have generalized the problem to different types of applications on FPGA, and created several design spaces in order to improve our understanding of how to efficiently explore these spaces without wasting design time. By using the OpenCL language for hardware design, we have established some correlation between optimization goals on FPGA and other computing platforms. We can leverage these correlations by organizing the design spaces into clusters to further prune non-optimal designs. These results can lead to more research into transferring knowledge between design spaces to improve the convergence of design space exploration

algorithms. Finally, we have designed a new active learning-based design space exploration platform that can reach the Pareto front with a very low amount of designs evaluated. We used a combination of properly chosen regression models along with a sampling strategy to find the optimal parameters on multiple axes. The choice of regression model can impact the design space exploration, but we can smartly choose a model based on simple reinforcement learning strategies. These design space exploration frameworks can help build complex systems by finding the set of optimal parameters, while minimizing the number of designs to evaluate. We have tested our framework on hardware designs and SLAM software applications successfully, suggesting that such a framework could be used to speed up the design and testing of a complex 3D scanning system.

Bibliography

[1] Atne source code. https://github.com/qkgautier/ATNE.

[2] Infinitam on fpga: Kernel source and results. https://github.com/fpga3d/infinitam_fpga.

[3] Intel fpga sdk for opencl (altera opencl sdk). https://www.intel.com/content/www/us/en/software/programmable/sdk-for-opencl/overview.html.

[4] Spector repository. https://github.com/KastnerRG/spector.

[5] Xilinx sdaccel. http://www.xilinx.com/products/design-tools/software-zone/sdaccel.html.

[6] Abiel Aguilar-González, Miguel Arias-Estrada, and François Berry. Robust feature extraction algorithm suitable for real-time embedded applications. *Journal of Real-Time Image Processing*, June 2017.

[7] Krste Asanovic, Ras Bodik, Bryan Christopher Catanzaro, Joseph James Gebis, Parry Husbands, Kurt Keutzer, David A Patterson, William Lester Plishker, John Shalf, Samuel Webb Williams, et al. The landscape of parallel computing research: A view from berkeley. Technical report, Technical Report UCB/EECS-2006-183, EECS Department, University of , 2006.

[8] David F. Bacon, Rodric Rabbah, and Sunil Shukla. Fpga programming for the masses. *Commun. ACM*, 56(4):56–63, April 2013.

[9] P. J. Besl and N. D. McKay. A method for registration of 3-d shapes. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 14(2):239–256, February 1992.

[10] Hans-Georg Beyer and Bernhard Sendhoff. Robust optimization a comprehensive survey. *Computer Methods in Applied Mechanics and Engineering*, 196(33):3190–3218, 2007.

[11] B. Bodin, L. Nardi, M. Z. Zia, H. Wagstaff, G. S. Shenoy, M. Emani, J. Mawer, C. Kotselidis, A. Nisbet, M. Lujan, B. Franke, P. H. J. Kelly, and M. O'Boyle. Integrating algorithmic parameters into benchmarking and design space exploration in 3d scene understanding. In *2016 International Conference on Parallel Architecture and Compilation Techniques (PACT)*, pages 57–69, September 2016.

[12] Bruno Bodin, Harry Wagstaff, Sajad Saeedi, Luigi Nardi, Emanuele Vespa, John H Mayer, Andy Nisbet, Mikel Lujn, Steve Furber, Andrew J Davison, Paul H.J. Kelly, and Michael O'Boyle. Slambench2: Multi-objective head-to-head benchmarking for visual slam. In *IEEE Intl. Conf. on Robotics and Automation (ICRA)*, May 2018.

[13] K. Boikos and C. S. Bouganis. Semi-dense slam on an fpga soc. In *2016 26th International Conference on Field Programmable Logic and Applications (FPL)*, pages 1–4, August 2016.

[14] K. Boikos and C. S. Bouganis. A high-performance system-on-chip architecture for direct tracking for slam. In *2017 27th International Conference on Field Programmable Logic and Applications (FPL)*, pages 1–7, September 2017.

[15] Leo Breiman. Random forests. *Machine Learning*, 45(1):5–32, October 2001.

[16] G. Bresson, Z. Alsayed, L. Yu, and S. Glaser. Simultaneous localization and mapping: A survey of current trends in autonomous driving. *IEEE Transactions on Intelligent Vehicles*, 2(3):194–220, September 2017.

[17] Mihai Bujanca, Paul Gafton, Sajad Saeedi, Andy Nisbet, Bruno Bodin, MF O'Boyle, Andrew J Davison, PH Kelly, Graham Riley, Barry Lennox, et al. Slambench 3.0: Systematic automated reproducible evaluation of slam systems for robot vision challenges and scene understanding. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2019.

[18] W. Burgard, C. Stachniss, G. Grisetti, B. Steder, R. Kmmerle, C. Dornhege, M. Ruhnke, A. Kleiner, and J. D. Tards. A comparison of slam algorithms based on a graph of relations. In *2009 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 2089–2095, October 2009.

[19] Marcello A. Canuto, Francisco Estrada-Belli, Thomas G. Garrison, Stephen D. Houston, Mary Jane Acuña, Milan Kováč, Damien Marken, Philippe Nondédéo, Luke Auld-Thomas, Cyril Castanet, David Chatelain, Carlos R. Chiriboga, Tomáš Drápela, Tibor Lieskovský, Alexandre Tokovinine, Antolín Velasquez, Juan C. Fernández-Díaz, and Ramesh Shrestha. Ancient lowland maya complexity as revealed by airborne laser scanning of northern guatemala. *Science*, 361(6409), 2018.

[20] Senthil K. Chandrasegaran, Karthik Ramani, Ram D. Sriram, Imré HorváTh, Alain Bernard, Ramy F. Harik, and Wei Gao. The evolution, challenges, and future of knowledge representation in product design systems. *Comput. Aided Des.*, 45(2):204–228, February 2013.

[21] Daniel W. Chang, Christipher D. Jenkins, Philip C. Garcia, Syed Z. Gilani, Paula Aguilera, Aishwarya Nagarajan, Michael J. Anderson, Matthew A. Kenny, Sean M. Bauer, Michael J. Schulte, and Katherine Compton. ERCBench: An open-source benchmark suite for embedded and reconfigurable computing. *Proceedings - 2010 International Conference on Field Programmable Logic and Applications, FPL 2010*, pages 408–413, 2010.

[22] Olivier Chapelle and Lihong Li. An empirical evaluation of thompson sampling. In J. Shawe-Taylor, R. S. Zemel, P. L. Bartlett, F. Pereira, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 24*, pages 2249–2257. Curran Associates, Inc., 2011.

[23] Arlen F. Chase, Diane Z. Chase, Jaime J. Awe, John F. Weishampel, Gyles Iannone, Holley Moyes, Jason Yaeger, and M. Kathryn Brown. The use of lidar in understanding the ancient maya landscape: Caracol and western belize. *Advances in Archaeological Practice*, 2(3):208–221, 2014.

[24] S. Che, M. Boyer, J. Meng, D. Tarjan, J. W. Sheaffer, S. H. Lee, and K. Skadron. Rodinia: A benchmark suite for heterogeneous computing. In *Workload Characterization, 2009. IISWC 2009. IEEE International Symposium on*, pages 44–54, October 2009.

[25] Paolo Cignoni, Claudio Rocchini, and Roberto Scopigno. Metro: Measuring error on simplified surfaces. In *Computer Graphics Forum*, volume 17, pages 167–174. Wiley Online Library, 1998.

[26] Carlos A Coello Coello, Gary B Lamont, David A Van Veldhuizen, et al. Multiobjective evolutionary algorithms: A survey of the state of the art. *Swarm and Evolutionary Computation*, 1(1):32–49, 2011.

[27] Gaspar Muñoz Cosme, Cristina Vidal Lorenzo, and Alessandro Merlo. La acrópolis de chilonché (guatemala): Crónica de las investigaciones de un patrimonio en riesgo en el área maya. *Restauro Archeologico*, 22(2):99–115, 2014.

[28] Xiuhai Cui, Datong Liu, Yu Peng, and Xiyuan Peng. Estimating the circuit delay of fpga with a transfer learning method. In *AOPC 2017: 3D Measurement Technology for Intelligent Manufacturing*, volume 10458, pages 10458 – 10458 – 6, 2017.

[29] Brian Curless and Marc Levoy. A volumetric method for building complex models from range images. In *Proceedings of the 23rd Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '96, pages 303–312, New York, NY, USA, 1996. ACM.

[30] Christian Damgaard and Jacob Weiner. Describing inequality in plant size or fecundity. *Ecology*, 81(4):1139–1142, 2000.

[31] A. J. Davison, I. D. Reid, N. D. Molton, and O. Stasse. Monoslam: Real-time single camera slam. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 29(6):1052–1067, June 2007.

[32] G. De Michell and R. K. Gupta. Hardware/software co-design. *Proceedings of the IEEE*, 85(3):349–365, March 1997.

[33] Andrea Di Filippo, Luis Javier Snchez-Aparicio, Salvatore Barba, Jos Antonio Martn-Jimnez, Roco Mora, and Diego Gonzlez Aguilera. Use of a wearable mobile laser system

in seamless indoor 3d mapping of a complex historical site. *Remote Sensing*, 10(12), 2018.

[34] A. C. Duarte, G. B. Zaffari, R. T. S. da Rosa, L. M. Longaray, P. Drews, and S. S. C. Botelho. Towards comparison of underwater slam methods: An open dataset collection. In *OCEANS 2016 MTS/IEEE Monterey*, pages 1–5, September 2016.

[35] H. Durrant-Whyte and T. Bailey. Simultaneous localization and mapping: part i. *IEEE Robotics Automation Magazine*, 13(2):99–110, June 2006.

[36] EM Farella. 3d mapping of underground environments with a hand-held laser scanner. In *Proc. SIFET annual conference*, 2016.

[37] Francesco Fassi, Luigi Fregonese, Sebastiano Ackermann, and V De Troia. Comparison between laser scanning and automated 3d modelling techniques to reconstruct complex and extensive cultural heritage areas. volume XL-5/W1, February 2013.

[38] Wu-chun Feng, Heshan Lin, Thomas Scogland, and Jing Zhang. Opencl and the 13 dwarfs: A work in progress. In *Proceedings of the 3rd ACM/SPEC International Conference on Performance Engineering*, ICPE '12, pages 291–294, New York, NY, USA, 2012. ACM.

[39] C. Forster, Z. Zhang, M. Gassner, M. Werlberger, and D. Scaramuzza. Svo: Semidirect visual odometry for monocular and multicamera systems. *IEEE Transactions on Robotics*, 33(2):249–265, April 2017.

[40] Thomas G. Garrison, Jose Luis Garrido Lpez, and Alyce de Carteret. Investigaciones en la estructura m7-1 (operacin 21) (pirmide del dintel de madera). *Proyecto Arqueolgico El Zotz Informe No. 7, Temporada 2012*, pages 59 – 97, 2012.

[41] Thomas G. Garrison, Dustin Richmond, Perry Naughton, Eric Lo, Sabrina Trinh, Zachary Barnes, Albert Lin, Curt Schurgers, Ryan Kastner, Sarah E. Newman, and et al. Tunnel vision: Documenting excavations in three dimensions with lidar technology. *Advances in Archaeological Practice*, 4(2):192–204, 2016.

[42] Q. Gautier, A. Althoff, P. Meng, and R. Kastner. Spector: An opencl fpga benchmark suite. In *2016 International Conference on Field Programmable Technology (FPT)*, pages 141–148, December 2016.

[43] Q. Gautier, A. Shearer, J. Matai, D. Richmond, P. Meng, and R. Kastner. Real-time 3d reconstruction for fpgas: A case study for evaluating the performance, area, and programmability trade-offs of the altera opencl sdk. In *2014 International Conference on Field-Programmable Technology (FPT)*, pages 326–329, December 2014.

[44] Quentin Gautier, Alric Althoff, and Ryan Kastner. Fpga architectures for real-time dense slam. In *2019 IEEE 30th International Conference on Application-specific Systems, Architectures and Processors (ASAP)*. IEEE, 2019.

[45] A. Geiger, P. Lenz, and R. Urtasun. Are we ready for autonomous driving? the kitti vision benchmark suite. In *2012 IEEE Conference on Computer Vision and Pattern Recognition*, pages 3354–3361, June 2012.

[46] Corrado Gini. Variabilità e mutabilità. *Reprinted in Memorie di metodologica statistica (Ed. Pizetti E, Salvemini, T). Rome: Libreria Eredi Virgilio Veschi*, 1, 1912.

[47] Daniel Girardeau-Montaut. Cloudcompare. 2011.

[48] Ryan Haney, Theresa Meuse, Jeremy Kepner, and James Lebak. The hpec challenge benchmark suite. In *HPEC 2005 Workshop*, 2005.

[49] Robert M. Haralick. Performance characterization in computer vision. *CVGIP: Image Underst.*, 60(2):245–249, September 1994.

[50] Steven P. Haveman and G. Maarten Bonnema. Requirements for high level models supporting design space exploration in model-based systems engineering. *Procedia Computer Science*, 16:293–302, 2013. 2013 Conference on Systems Engineering Research.

[51] Cato Holler. Chapter 101 - pseudokarst. In William B. White, David C. Culver, and Tanja Pipan, editors, *Encyclopedia of Caves (Third Edition)*, pages 836–849. Academic Press, third edition edition, 2019.

[52] Stephen D Houston, Sarah Newman, Edwin Román, and Thomas Garrison. A tomb and its setting. In *Temple of the Night Sun: a royal tomb at El Diablo, Guatemala*, pages 12–29. Precolumbia Mesoweb Press, 2015.

[53] A. Huletski, D. Kartashov, and K. Krinkin. Evaluation of the modern visual slam methods. In *2015 Artificial Intelligence and Natural Language and Information Extraction, Social Media and Web Search FRUCT Conference (AINL-ISMW FRUCT)*, pages 19–25, November 2015.

[54] I. Z. Ibragimov and I. M. Afanasyev. Comparison of ros-based visual slam methods in homogeneous indoor environment. In *2017 14th Workshop on Positioning, Navigation and Communications (WPNC)*, pages 1–6, October 2017.

[55] Z. István, G. Alonso, M. Blott, and K. Vissers. A flexible hash table design for 10gbps key-value stores on fpgas. In *2013 23rd International Conference on Field programmable Logic and Applications*, pages 1–8, September 2013.

[56] Shahram Izadi, David Kim, Otmar Hilliges, David Molyneaux, Richard Newcombe, Pushmeet Kohli, Jamie Shotton, Steve Hodges, Dustin Freeman, Andrew Davison, and Andrew Fitzgibbon. Kinectfusion: Real-time 3d reconstruction and interaction using a moving depth camera. In *Proceedings of the 24th Annual ACM Symposium on User Interface Software and Technology*, UIST '11, pages 559–568, New York, NY, USA, 2011. ACM.

[57] Matthew Johnson-Roberson, Oscar Pizarro, Stefan B. Williams, and Ian Mahon. Generation and visualization of large-scale three-dimensional reconstructions from underwater robotic surveys. *Journal of Field Robotics*, 27(1):21–51, 2010.

[58] Olaf Kahler, Victor Adrian Prisacariu, Carl Yuheng Ren, Xin Sun, Philip Torr, and David Murray. Very high frame rate volumetric integration of depth images on mobile devices. *IEEE Transactions on Visualization and Computer Graphics*, 21(11):1241–1250, November 2015.

[59] Olaf Kähler, Victor A. Prisacariu, and David W. Murray. Real-time large-scale dense 3d reconstruction with loop closure. In Bastian Leibe, Jiri Matas, Nicu Sebe, and Max Welling, editors, *Computer Vision – ECCV 2016*, pages 500–516, Cham, 2016. Springer International Publishing.

[60] Eunsuk Kang, Ethan Jackson, and Wolfram Schulte. An approach for effective design space exploration. In Radu Calinescu and Ethan Jackson, editors, *Foundations of Computer Software. Modeling, Development, and Verification of Adaptive Systems*, pages 33–54, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg.

[61] Stefan Kohlbrecher, Oskar Von Stryk, Johannes Meyer, and Uwe Klingauf. A flexible and scalable slam system with full 3d motion estimation. In *2011 IEEE International Symposium on Safety, Security, and Rescue Robotics*, pages 155–160. IEEE, 2011.

[62] Marek Kraft, Michał Nowicki, Adam Schmidt, Michał Fularz, and Piotr Skrzypczyński. Toward evaluation of visual navigation algorithms on rgb-d data from the first- and second-generation kinect. *Machine Vision and Applications*, 28(1):61–74, February 2017.

[63] Konstantinos Krommydas, Wu chun Feng, Muhsen Owaida, Christos D. Antonopoulos, and Nikolaos Bellas. On the characterization of OpenCL dwarfs on fixed and reconfigurable platforms. *Proceedings of the International Conference on Application-Specific Systems, Architectures and Processors*, pages 153–160, 2014.

[64] Sunil L Kukreja, Johan Löfberg, and Martin J Brenner. A least absolute shrinkage and selection operator (lasso) for nonlinear system identification. *IFAC Proceedings Volumes*, 39(1):814–819, 2006.

[65] M. Labb and F. Michaud. Memory management for real-time appearance-based loop closure detection. In *2011 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 1271–1276, September 2011.

[66] M. Labb and F. Michaud. Appearance-based loop closure detection for online large-scale and long-term operation. *IEEE Transactions on Robotics*, 29(3):734–745, June 2013.

[67] M. Labb and F. Michaud. Online global loop closure detection for large-scale multi-session graph-based slam. In *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 2661–2666, September 2014.

[68] Butler W. Lampson. Hints for computer system design. *SIGOPS Oper. Syst. Rev.*, 17(5):33–48, October 1983.

[69] Craig Larman and Victor R. Basili. Iterative and incremental development: A brief history. *Computer*, 36(6):47–56, June 2003.

[70] M M. Lehman, J F. Ramil, P D. Wernick, D E. Perry, and W M. Turski. Metrics and laws of software evolution - the nineties view. In *Proceedings of the 4th International Symposium on Software Metrics*, METRICS '97, pages 20–, Washington, DC, USA, 1997. IEEE Computer Society.

[71] Max Leingartner, Johannes Maurer, Alexander Ferrein, and Gerald Steinbauer. Evaluation of sensors and mapping approaches for disasters in tunnels. *Journal of Field Robotics*, 33(8):1037–1057, 2016.

[72] R. Li, J. Liu, L. Zhang, and Y. Hang. Lidar/mems imu integrated navigation (slam) method for a small uav in indoor environments. In *2014 DGON Inertial Sensors and Systems (ISS)*, pages 1–15, September 2014.

[73] Dong Liu and Benjamin Carrion Schafer. Efficient and reliable high-level synthesis design space explorer for fpgas. In *Field Programmable Logic and Applications (FPL), 2016 26th International Conference on*, pages 1–8. IEEE, 2016.

[74] Hung-Yi Liu and L. P. Carloni. On learning-based methods for design-space exploration with high-level synthesis. In *Design Automation Conference (DAC), 2013 50th ACM/EDAC/IEEE*, DAC '13, pages 1–7, New York, NY, USA, May 2013. ACM.

[75] G. Mariani, G. Palermo, V. Zaccaria, and C. Silvano. Desperate++: An enhanced design space exploration framework using predictive simulation scheduling. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 34(2):293–306, February 2015.

[76] A. Masiero, F. Fissore, A. Guarnieri, M. Piragnolo, and A. Vettore. Comparison of Low Cost Photogrammetric Survey with Tls and Leica Pegasus Backpack 3d Modelss. *ISPRS - International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, pages 147–153, November 2017.

[77] P. Meng, A. Althoff, Q. Gautier, and R. Kastner. Adaptive threshold non-pareto elimination: Re-thinking machine learning for system level design space exploration on fpgas. In *2016 Design, Automation Test in Europe Conference Exhibition (DATE)*, pages 918–923, March 2016.

[78] Johannes Meyer, Paul Schnitzspan, Stefan Kohlbrecher, Karen Petersen, Mykhaylo Andriluka, Oliver Schwahn, Uwe Klingauf, Stefan Roth, Bernt Schiele, and Oskar von Stryk. A semantic world model for urban search and rescue based on heterogeneous sensors. In Javier Ruiz-del Solar, Eric Chown, and Paul G. Plöger, editors, *RoboCup 2010: Robot Soccer World Cup XIV*, pages 180–193, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg.

[79] Grigorios Mingas, Emmanouil Tsardoulias, and Loukas Petrou. An fpga implementation of the smg-slam algorithm. *Microprocess. Microsyst.*, 36(3):190–204, May 2012.

[80] Vipul Kumar Mishra and Anirban Sengupta. Mo-pse: Adaptive multi-objective particle swarm optimization based design space exploration in architectural synthesis for application specific processor design. *Advances in Engineering Software*, 67:111–124, 2014.

[81] Caitlin T. Mueller and John A. Ochsendorf. Combining structural performance and designer preferences in evolutionary design space exploration. *Automation in Construction*, 52:70–82, 2015.

[82] R. Mur-Artal and J. D. Tards. Orb-slam2: An open-source slam system for monocular, stereo, and rgb-d cameras. *IEEE Transactions on Robotics*, 33(5):1255–1262, October 2017.

[83] K. E. Murray, S. Whitty, S. Liu, J. Luu, and V. Betz. Titan: Enabling large and complex benchmarks in academic cad. In *2013 23rd International Conference on Field programmable Logic and Applications*, pages 1–8, September 2013.

[84] V. Nair, Z. Yu, T. Menzies, N. Siegmund, and S. Apel. Finding faster configurations using flash. *IEEE Transactions on Software Engineering*, pages 1–1, 2018.

[85] L. Nardi, B. Bodin, S. Saeedi, E. Vespa, A. J. Davison, and P. H. J. Kelly. Algorithmic performance-accuracy trade-off in 3d vision applications using hypermapper. In *2017 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, pages 1434–1443, May 2017.

[86] R. A. Newcombe, S. Izadi, O. Hilliges, D. Molyneaux, D. Kim, A. J. Davison, P. Kohi, J. Shotton, S. Hodges, and A. Fitzgibbon. Kinectfusion: Real-time dense surface mapping and tracking. In *2011 10th IEEE International Symposium on Mixed and Augmented Reality*, pages 127–136, October 2011.

[87] Matthias Niessner, Michael Zollhöfer, Shahram Izadi, and Marc Stamminger. Real-time 3d reconstruction at scale using voxel hashing. *ACM Trans. Graph.*, 32(6):169:1–169:11, November 2013.

[88] J. Nikolic, J. Rehder, M. Burri, P. Gohl, S. Leutenegger, P. T. Furgale, and R. Siegwart. A synchronized visual-inertial sensor system with fpga pre-processing for accurate real-time slam. In *2014 IEEE International Conference on Robotics and Automation (ICRA)*, pages 431–437, May 2014.

[89] Erica Nocerino, Fabio Menna, Fabio Remondino, Isabella Toschi, and Pablo Rodrguez-Gonzlvez. Investigation of indoor and outdoor performance of two portable mobile mapping systems. In Fabio Remondino and Mark R. Shortis, editors, *Videometrics, Range Imaging, and Applications XIV*, volume 10332, pages 10332 – 10332 – 15. SPIE, June 2017.

[90] Anton Obukhov and Alexander Kharlamov. Discrete cosine transform for 8x8 blocks with cuda, 2008.

[91] G. Palermo, C. Silvano, and V. Zaccaria. Discrete particle swarm optimization for multi-objective design space exploration. In *2008 11th EUROMICRO Conference on Digital System Design Architectures, Methods and Tools*, pages 641–644, September 2008.

[92] G. Palermo, C. Silvano, and V. Zaccaria. Respir: A response surface-based pareto iterative refinement for application-specific design space exploration. *TCAD*, pages 1816–1829, 2009.

[93] Maurizio Palesi and Tony Givargis. Multi-objective design space exploration using genetic algorithms. In *Proceedings of the Tenth International Symposium on Hardware/Software Codesign*, CODES '02, pages 67–72, New York, NY, USA, 2002. ACM.

[94] Czyzak Piotr and Jaszkiewicz Adrezej. Pareto simulated annealinga metaheuristic technique for multiple-objective combinatorial optimization. *Journal of Multi-Criteria Decision Analysis*, 7(1):34–47, 1998.

[95] Piyush Rai, Avishek Saha, Hal Daumé III, and Suresh Venkatasubramanian. Domain adaptation meets active learning. In *Proceedings of the NAACL HLT 2010 Workshop on Active Learning for Natural Language Processing*, pages 27–32. Association for Computational Linguistics, 2010.

[96] Rajkumar Roy, Srichand Hinduja, and Roberto Teti. Recent advances in engineering design optimisation: Challenges and future trends. *CIRP Annals*, 57(2):697–715, 2008.

[97] Radu Bogdan Rusu and Steve Cousins. 3d is here: Point cloud library (pcl). In *2011 IEEE International Conference on Robotics and Automation*, pages 1–4, Shanghai, China, May 2011.

[98] B. C. Schafer and A. Mahapatra. S2cbench: Synthesizable systemc benchmark suite for high-level synthesis. *IEEE Embedded Systems Letters*, 6(3):53–56, September 2014.

[99] Benjamin Carrion Schafer. Probabilistic multiknob high-level synthesis design space exploration acceleration. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 35(3):394–406, 2016.

[100] Benjamin Carrion Schafer. Parallel high-level synthesis design space exploration for behavioral ips of exact latencies. *ACM Trans. Des. Autom. Electron. Syst.*, 22(4), May 2017.

[101] Bernhard Schätz, Sebastian Voss, and Sergey Zverlov. Automating design-space exploration: Optimal deployment of automotive sw-components in an iso26262 context. In *Proceedings of the 52Nd Annual Design Automation Conference*, DAC '15, pages 99:1–99:6, New York, NY, USA, 2015. ACM.

[102] Burr Settles. Active Learning. *Synthesis Lectures on Artificial Intelligence and Machine Learning*, 6(1):1–114, 2012.

[103] Biruk G. Sileshi, Juan Oliver, R. Toledo, Jose Gonçalves, and Pedro Costa. Particle filter slam on fpga: A case study on robot@factory competition. In Luís Paulo Reis, António Paulo Moreira, Pedro U. Lima, Luis Montano, and Victor Muñoz-Martinez, editors, *Robot 2015: Second Iberian Robotics Conference*, pages 411–423, Cham, 2016. Springer International Publishing.

[104] Randall C. Smith and Peter Cheeseman. On the representation and estimation of spatial uncertainty. *The International Journal of Robotics Research*, 5(4):56–68, 1986.

[105] J. Sturm, N. Engelhard, F. Endres, W. Burgard, and D. Cremers. A benchmark for the evaluation of rgb-d slam systems. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 573–580, October 2012.

[106] Keisuke Tateno, Federico Tombari, Iro Laina, and Nassir Navab. Cnn-slam: Real-time dense monocular slam with learned depth prediction. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, July 2017.

[107] D. T. Tertei, J. Piat, and M. Devy. Fpga design and implementation of a matrix multiplier based accelerator for 3d ekf slam. In *2014 International Conference on ReConFigurable Computing and FPGAs (ReConFig14)*, pages 1–6, December 2014.

[108] William R. Thompson. On the likelihood that one unknown probability exceeds another in view of the evidence of two samples. *Biometrika*, 25(3/4):285–294, 1933.

[109] Robert Tibshirani. Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society. Series B (Methodological)*, 58(1):267–288, 1996.

[110] D. Tong, S. Zhou, and V. K. Prasanna. High-throughput online hash table on fpga. In *2015 IEEE International Parallel and Distributed Processing Symposium Workshop*, pages 105–112, May 2015.

[111] O. Ulusel, C. Picardo, C. B. Harris, S. Reda, and R. I. Bahar. Hardware acceleration of feature detection and description algorithms on low-power embedded platforms. In *2016 26th International Conference on Field Programmable Logic and Applications (FPL)*, pages 1–9, August 2016.

[112] M Mitchell Waldrop. The chips are down for moores law. *Nature News*, 530(7589):144, 2016.

[113] C. Wang, L. Gong, Q. Yu, X. Li, Y. Xie, and X. Zhou. Dlau: A scalable deep learning accelerator unit on fpga. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 36(3):513–517, March 2017.

[114] Long Wang, Tong-guang Wang, and Yuan Luo. Improved non-dominated sorting genetic algorithm (nsga)-ii in multi-objective optimization studies of wind turbine blades. *Applied Mathematics and Mechanics*, 32(6):739–748, June 2011.

[115] S. Wang, Y. Liang, and Wei Zhang. Flexcl: An analytical performance model for opencl workloads on flexible fpgas. In *2017 54th ACM/EDAC/IEEE Design Automation Conference (DAC)*, pages 1–6, June 2017.

[116] Ying Wang, Jie Xu, Yinhe Han, Huawei Li, and Xiaowei Li. Deepburning: Automatic generation of fpga-based learning accelerators for the neural network family. In *Proceedings of the 53rd Annual Design Automation Conference*, DAC '16, pages 110:1–110:6, New York, NY, USA, 2016. ACM.

[117] Z Wang, B He, W Zhang, and S Jiang. A performance analysis framework for optimizing opencl applications on fpgas. *2016 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pages 114–125, 2016.

[118] Thomas Whelan, Stefan Leutenegger, R Salas-Moreno, Ben Glocker, and Andrew Davison. Elasticfusion: Dense slam without a pose graph, 2015.

[119] Chang Xu, Gai Liu, Ritchie Zhao, Stephen Yang, Guojie Luo, and Zhiru Zhang. A parallel bandit-based approach for autotuning fpga compilation. In *Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, FPGA '17, pages 157–166, New York, NY, USA, 2017. ACM.

[120] Naci Yastikli. Documentation of cultural heritage using digital photogrammetry and laser scanning. *Journal of Cultural Heritage*, 8(4):423–427, 2007.

[121] Kai Yu, Jinbo Bi, and Volker Tresp. Active learning via transductive experimental design. In *Proceedings of the 23rd international conference on Machine learning*, pages 1081–1088. ACM, 2006.

[122] S. Zennaro, M. Munaro, S. Milani, P. Zanuttigh, A. Bernardi, S. Ghidoni, and E. Menegatti. Performance evaluation of the 1st and 2nd generation kinect for multimedia applications. In *2015 IEEE International Conference on Multimedia and Expo (ICME)*, pages 1–6, June 2015.

[123] Chen Zhang, Peng Li, Guangyu Sun, Yijin Guan, Bingjun Xiao, and Jason Cong. Optimizing fpga-based accelerator design for deep convolutional neural networks. In *Proceedings of the 2015 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, FPGA '15, pages 161–170, New York, NY, USA, 2015. ACM.

[124] Hui Zhang, Jason E. Fritts, and Sally A. Goldman. Image segmentation evaluation: A survey of unsupervised methods. *Comput. Vis. Image Underst.*, 110(2):260–280, May 2008.

[125] Yu Jin Zhang. A survey on evaluation methods for image segmentation. *Pattern recognition*, 29(8):1335–1346, 1996.

[126] G. Zhong, A. Prakash, S. Wang, Y. Liang, T. Mitra, and S. Niar. Design space exploration of fpga-based accelerators with multi-level parallelism. In *Design, Automation Test in Europe Conference Exhibition (DATE), 2017*, pages 1141–1146, March 2017.

[127] Guanwen Zhong, Alok Prakash, Yun Liang, Tulika Mitra, and Smail Niar. Lin-analyzer: A high-level performance analysis tool for fpga-based accelerators. In *Proceedings of the 53rd Annual Design Automation Conference*, DAC '16, New York, NY, USA, 2016. ACM.

[128] Huizhong Zhou, Benjamin Ummenhofer, and Thomas Brox. Deeptam: Deep tracking and mapping. In *The European Conference on Computer Vision (ECCV)*, September 2018.

[129] Robert Zlot, Michael Bosse, Kelly Greenop, Zbigniew Jarzab, Emily Juckes, and Jonathan Roberts. Efficiently capturing large, complex cultural heritage sites with a handheld mobile 3d laser mapping system. *Journal of Cultural Heritage*, 15(6):670–678, 2014.

[130] Marcela Zuluaga, Andreas Krause, Peter Milder, and Markus Püschel. "smart" design space sampling to predict pareto-optimal solutions. In *Proceedings of the 13th ACM SIGPLAN/SIGBED International Conference on Languages, Compilers, Tools and Theory for Embedded Systems*, LCTES '12, pages 119–128, New York, NY, USA, 2012. ACM.

[131] Marcela Zuluaga, Andreas Krause, and Markus Püschel. ε-pal: An active learning approach to the multi-objective optimization problem. *J. Mach. Learn. Res.*, 17(1):3619–3650, January 2016.

[132] Marcela Zuluaga, Guillaume Sergent, Andreas Krause, and Markus Püschel. Active learning for multi-objective optimization. In *Proceedings of the 30th International Conference on Machine Learning*, pages 462–470, 2013.