Integrating Algorithmic Parameters into Benchmarking and Design Space Exploration in 3D Scene Understanding

Bruno Bodin
bbodin@inf.ed.ac.uk

Luigi Nardi
l.nardi@imperial.ac.uk

M. Zeeshan Zia
zeeshan.zia@imperial.ac.uk

Harry Wagstaff
h.wagstaff@inf.ed.ac.uk

Govind Sreekar Shenoy
gsreekar@inf.ed.ac.uk

Murali Emani
emani1@llnl.gov

John Mawer
john.mawer@manchester.ac.uk

Christos Kotselidis
christos.kotselidis@manchester.ac.uk

Andy Nisbet
andy.nisbet@manchester.ac.uk

Mikel Lujan
mikel.lujan@manchester.ac.uk

Björn Franke
bfranke@inf.ed.ac.uk

Paul H. J. Kelly
p.kelly@imperial.ac.uk

Michael O'Boyle
mob@inf.ed.ac.uk

ABSTRACT

System designers typically use well-studied benchmarks to evaluate and improve new architectures and compilers. We design tomorrow's systems based on yesterday's applications. In this paper we investigate an emerging application, 3D scene understanding, likely to be significant in the mobile space in the near future. Until now, this application could only run in real-time on desktop GPUs. In this work, we examine how it can be mapped to power constrained embedded systems. Key to our approach is the idea of incremental co-design exploration, where optimization choices that concern the domain layer are incrementally explored together with low-level compiler and architecture choices. The goal of this exploration is to reduce execution time while minimizing power and meeting our quality of result objective. As the design space is too large to exhaustively evaluate, we use active learning based on a random forest predictor to find good designs. We show that our approach can, for the first time, achieve dense 3D mapping and tracking in the real-time range within a 1W power budget on a popular embedded device. This is a 4.8x execution time improvement and a 2.8x power reduction compared to the state-of-the-art.

Keywords

design space exploration; DSE; computer vision; SLAM; embedded systems

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

PACT '16, September 11 - 15, 2016, Haifa, Israel

© 2016 Copyright held by the owner/author(s). Publication rights licensed to ACM.

ISBN 978-1-4503-4121-9/16/09…$15.00

DOI: http://dx.doi.org/10.1145/2967938.2967963

1. INTRODUCTION

The computing landscape has changed dramatically over the last decade. We have witnessed the decline of desktops and the rise of mobile devices as computing platforms. At the system level, power constraints have caused a fundamental shift to parallel heterogeneous platforms which is particularly important in thermally limited embedded mobile devices.

More recently, the well-known dark silicon challenge suggests that we will not be able to simultaneously power on all the cores (or transistors) on a device [17]. Heterogeneous multi-core systems have emerged as a promising solution to this problem. For example, the ARM big.LITTLE technology [9] puts both a high power, high performance core cluster, and a more efficient but less computationally powerful cluster, on the same die. Software can then partition between these cores, or switch off cores entirely, depending on requirements. By mapping different parts of an application onto the appropriate specialized hardware resource, we can use the available power budget in an optimal manner. For that reason, heterogeneous multi-processor system-on-chips (MPSoCs) have been widely adopted in mobile embedded systems.

In order to design and program heterogeneous MPSoCs, a vertical approach is necessary. Deep knowledge of all levels of the stack, from compilers to the micro-architecture, is needed in order to optimally map the executed code onto such diverse hardware resources. Additionally, deep domain knowledge may be required to tune software parameters to meet multiple conflicting design goals. This paper shows how we can go beyond conventional benchmarking in computer systems research by exposing the algorithmic-level design space.

Traditionally, system designers have evaluated new architecture and compiler features using well-studied and broadly accepted benchmarks such as SPEC2006 [22]. However, since such benchmarks represent a historical snapshot of applications, they are not representative of modern requirements. In contrast, to design tomorrow's systems, we need to consider new emerging applications from diverse domains.

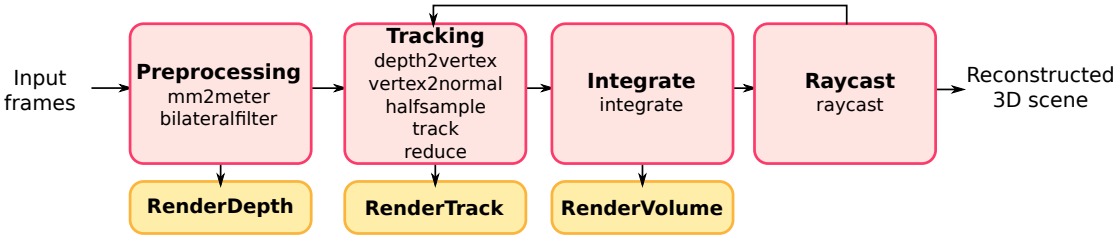


Figure 2: Key computational steps of the KFusion algorithm represented as a task graph. Each task comprises one or more OpenCL kernels as depicted.

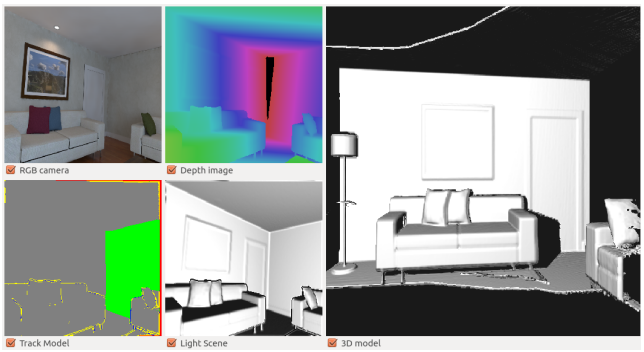


Figure 1: An RGB-D camera provides both RGB and depth information (top left and middle respectively); the depth frame is the input of the KFusion algorithm. Tracking results and reconstruction of the scene as seen from the current camera pose are shown in the bottom left and middle respectively. The reconstructed 3D scene from the initial pose is shown in the right; this is the output of KFusion.

In this paper we focus on one set of emerging applications that is becoming significant in the mobile space: real-time 3D scene understanding in computer vision. In particular, we investigate dense simultaneous localization and mapping (dense SLAM) algorithms which are extremely computationally demanding. One such dense SLAM algorithm is KinectFusion [33] (KFusion) which estimates the pose of a depth camera whilst constructing a highly detailed 3D model of the environment. Since such applications are typically tuned for high-end desktops with high power budget, executing them on power-constrained embedded devices is very challenging and, therefore, represents a realistic future application use case. We use the SLAMBench benchmarking framework [31], which contains a KFusion implementation, as it allows us to capture the performance metrics used to drive our design space exploration.

We explore the mapping of SLAM applications to power constrained heterogeneous platforms. The key element of our approach is the exploration of the mapping problem at multiple levels, vertically integrating the algorithmic domain and the implementation layers. Instead of ignoring levels of the computing stack, we perform co-design space exploration. In other words, we examine how algorithmic, compiler, and architecture configuration choices affect the performance of the underlying system. The rationale behind including the algorithmic parameters in the co-design space exploration is that although these algorithms are tuned for desktop systems, it is unlikely that the same configurations will be optimal in a mobile MPSoC setting.

We define the performance in terms of power consumption (measured in Watts, lower is better), accuracy of the computation (measured in centimeters, lower is better), and runtime (measured as wall clock time per frame in seconds, lower is better). The runtime is sometimes also quantified by the number of frames processed in one second, i.e. frames per second (FPS), higher is better; the current Microsoft Kinect (or equivalent ASUS Xtion Pro) RGB-D sensor runs at 30 FPS, so 30 FPS is needed for real-time processing. These three metrics interact and are considered simultaneously for a holistic evaluation of the system.

Since the co-design space can be extremely large, it is not feasible to try all possible configurations. Instead, we sample the domain space and automatically build a model that predicts the three performance metrics for a given configuration. Using this model, and a methodology from machine learning known as active learning, we predict a three dimensional performance Pareto curve that is then used to feed the lower level layers, driving the compiler and architecture parameter choices. By exploring the resulting Pareto curve we obtain a mapping to an embedded platform that results in a 6.6-fold speedup over the original mobile implementation. More precisely, this new configuration runs at nearly 40 FPS while maintaining an acceptable accuracy (under 5 cm localization error) and keeping power consumption under 2 Watts. The Pareto front contains many more configurations, allowing us to trade between runtime, power consumption, and accuracy, depending on our desired goals. For example, we can also find points which minimize power consumption (e.g., a configuration providing 11.92 FPS at 0.65W) or which optimize for execution time without exceeding a given power budget (29.09 FPS at less than 1W).

This paper demonstrates that our co-design space exploration tailors future applications to future power-constrained systems. The contributions of this paper are as follows:

- We perform a vertical co-design space exploration considering algorithmic, compiler, and hardware layers.
- We show that domain-specific knowledge can be used to trade off multiple optimization goals at an algorithmic level, before considering low-level implementation choices.
- We introduce an effective method to guide the optimization using multi-objective performance prediction based on random forest and active learning.
- In order to explore the potential for this approach we evaluate our methodology on an emerging SLAM

benchmarking framework which supports quantitative evaluation of solution accuracy, execution time and power consumption. We obtain a 6.6x best improvement in execution time or a 4.3x best reduction in power dissipation over an hand-tuned implementation by a SLAM domain expert.

2. BACKGROUND

Simultaneous localization and mapping (SLAM) systems aim to perform real-time localization and mapping "simultaneously" from a sensor moving through an unknown environment. Localization typically estimates the location and pose of the sensor with respect to a map which is extended as the sensor explores the environment. Dense SLAM systems in particular map entire 3D surfaces, as opposed to non-dense (feature-based) systems where maps are represented at the level of sparse point landmarks. Dense SLAM systems enable a mobile robot to perform path planning and collision avoidance, or an augmented reality (AR) system to render physically plausible animations at appropriate locations in the scene [35, 18]. Recent advances in computer vision have led to the development of real-time algorithms for dense SLAM such as KFusion [33]. Such algorithms estimate the pose of a depth camera while building a highly detailed 3D model of the environment (see Figure 1).

Such real-time 3D scene understanding capabilities can radically change the way robots interact with the world [18]. While classical feature-based SLAM techniques are now crossing into mainstream products via embedded implementations, such as Project Tango [4] and Dyson 360 Eye [2], *dense* SLAM algorithms with their high computational requirements are largely at the prototype stage on GPU-based PC or laptop platforms [33]. However, when running in an embedded context, it is not feasible to include a large GPU with high power and cooling requirements. While offloading to a remote machine is possible in some circumstances, this can introduce additional latency which makes it unsuitable for real-time situations such as augmented reality or UAV navigation applications.

KFusion registers and fuses the stream of measured noisy depth frames from a depth camera (such as Microsoft Kinect), as the scene is viewed from different viewpoints, into a clean 3D geometric map. While it is beyond the scope of this paper to go into the details of the KFusion algorithm, we briefly outline the key computational steps involved in Figure 2. SLAMBench provides multiple implementations of the KFusion algorithm. We use the OpenCL implementation, and execute each OpenCL kernel on the GPU of our target platforms.

KFusion normalizes each incoming depth frame and applies a bilateral filter (*Preprocessing*) to reduce noise. In the *Tracking* step, it computes a point cloud (with normals) for each pixel in the camera frame of reference and estimates the new 3D pose of the moving camera by registering this point cloud with the current global map using iterative closest point (ICP) [11]. Once the new camera pose has been estimated, the corresponding depth map is fused into the current 3D reconstruction (*Integration*). KFusion utilizes a voxel grid as the data structure to represent the map, employing a truncated signed distance function (TSDF) to represent 3D surfaces. The 3D surfaces are present at the zero crossings of the TSDF and can be recovered by a *Raycasting* step, which is also useful for visualizing the reconstruction.

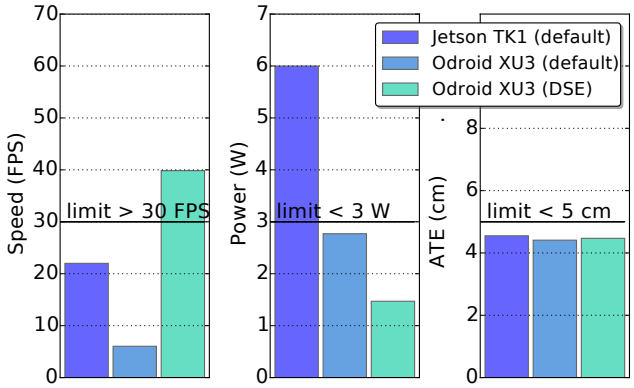


Figure 3: The three metrics (speed, power, and accuracy) of SLAMBench on Jetson TK1 and ODROID-XU3. The default configuration does not meet the real-time requirements of embedded systems in contrast to the DSE techniques introduced in this paper.

In this work we use the SLAMBench benchmarking framework [31] which enables evaluation of runtime, power consumption, and accuracy for KFusion. Figure 3 depicts the KFusion performance metric measurements for two different platforms, namely the NVIDIA Jetson TK1 featuring the Tegra K1 SoC and the ODROID-XU3 equipped with a Samsung Exynos 5422 SoC. For a mobile SLAM system to be usable, an implementation needs to provide real-time processing, i.e. a frame rate of 30 FPS for the common cameras, to consume less than 3W of power, which enables fan-less cooling, and to provide an absolute trajectory error (ATE) of at most 5 cm. The ATE is calculated as the mean difference between the real trajectory and the estimated trajectory of a camera produced by a SLAM implementation. Thus, smaller ATE implies less deviation from the real trajectory. We observe (Figure 3) that neither the NVIDIA Jetson TK1 nor the ODROID meet these requirements with the default configuration. While the speed of the TK1 implementation is close to 30 FPS it consumes significant power. The ODROID implementation meets the power constraint but its frame rate is too low. Note that both the platforms meet the accuracy constraint. The results on the ODROID platform after design space exploration (DSE) are also shown in Figure 3. The improved KFusion application now delivers FPS > 30 on the ODROID platform and, at the same time, consumes less power, thus, meeting both the performance and power constraints. Although the ATE has increased slightly, the design constraint is still satisfied.

This example demonstrates that for complex applications there is a trade-off between different performance metrics that can be exploited through an intelligent design space exploration.

3. METHODOLOGY

In this section we describe our approach, including a detailed explanation of the design space parameters, the objectives which we are targeting, and our incremental approach to exploring the design space. In Section 4, we go on to describe the search techniques we use to guide our exploration through the design space.

	Parameters	Values
Algorithmic	**Volume resolution**	64x64x64, 128x128x128
		256x256x256, 512x512x512
	μ distance	0.025, 0.075, 0.1, 0.2
	Pyramid level iterations	
	Level 1	3, 5, 7, 9, 11
	Level 2	3, 5, 7, 9, 11
	Level 3	3, 5, 7, 9, 11
	Compute size ratio	1, 2, 4, 8
	Tracking rate	1, 3, 5, 7, 9
	ICP threshold	0, 10^{-4}, 10^{-5}, 10^{-6}, 1
	Integration rate	1, 5, 10, 20, 30
Compiler	**OpenCL flags**	cl-mad-enable, cl-fast-relaxed-math, ...
	LLVM flags	O1, O2, O3, vectorize-slp-aggressive, ...
	Local work gr oup size	16, 32, 64, 96, 112, 128, 256
	Vectorization	
	Width	1, 2, 4, 8
	Direction	x, y
	Thread coarsening	
	Factor	1, 2, 4, 8, 16, 32
	Stride	1, 2, 4, 8, 16, 32
	Dimension	x, y
Architecture	**GPU processor frequency**	177, 266, 350, 420, 480, 543, 600, DVFS
	Number of active big cores	0, 1, 2, 3, 4
	Number of active little cores	1, 2, 3, 4

Table 1: The three co-design exploration spaces and the parameters used.

3.1 Experimental Setting

In order to evaluate our design space exploration (DSE) we use the SLAMBench framework with the ICL-NUIM [21, 20] dataset, specifically the first 400 frames of living room trajectory 2. We halved the original sequence in order to reduce the overall execution time of the benchmark; this was done after careful consideration that the accuracy metric is still representative of the whole sequence.

Usual approaches in performance optimization consider benchmark suites that are, in general, a set of small kernels extracted from real applications. A criticism to what can be learnt from a benchmark suite is that they may not well represent and capture the complex interaction of kernels in a real-world application. Our application is composed of more than 10 GPU-accelerated kernels. It presents the opportunity to tackle exploration of parameters at the algorithmic level that is not possible with conventional benchmark suites.

During execution the following three performance metrics are collected: 1) computation time, 2) absolute trajectory error (ATE) of the frame sequence, and 3) power consumption.

3.2 Co-Design Space

The possible values of the parameters taken into consideration for the co-design space exploration are summarized in Table 1. Here we look at three different spaces: algorithmic, compilation, and architecture.

Algorithmic Space.

In this paragraph we summarize the algorithmic parameters that mostly affect our performance metrics. In the case of the SLAMBench implementation of the KFusion algorithm, we have access to the listed parameters. An extensive explanation of these can be found in [33, 31].

- **Volume resolution**: The resolution of the scene being reconstructed. As an example, a 64x64x64 voxel grid captures less detail than a 256x256x256 voxel grid.
- **μ distance**: The output volume of KFusion is defined as a truncated signed distance function (TSDF) [33]. Every volume element (voxel) of the volume contains the best likelihood distance to the nearest visible surface, up to a truncation distance denoted by the parameter μ, also referred as mu in the text.
- **Pyramid level iterations**: The number of block averaging iterations to perform while building each level of the image pyramid.
- **Compute size ratio**: The fractional depth image resolution used as input. As an example, a value of 8 means that the raw frame is resized to one-eighth resolution.
- **Tracking rate**: The rate at which the KFusion algorithm attempts to perform localisation. A new localisation is performed after every tracking rate number of frames.
- **ICP threshold**: The threshold for the iterative closest point (ICP) algorithm [11] used during the tracking phase.
- **Integration rate**: As the output of KFusion is a volumetric representation of the recorded scene, it needs to be repeatedly expanded using new frames. A new frame is integrated after every integration rate number of frames.

We observe that the algorithmic design space consists of roughly 1,800,000 points. Furthermore, the exploration of algorithmic parameters involves trade-offs between accuracy, runtime, and power consumption.

Compiler Space.

In order to explore this space, we first compile each SLAMBench OpenCL kernel to LLVM IR using the clang compiler, before performing the selected LLVM optimization passes listed below. We then use Axtor [30] to produce OpenCL code from the processed LLVM IR. The optimized kernels are then used in SLAMBench instead of the original ones. A large number of compilation parameters exist, we selected those listed in Table 1, which are detailed below.

- **OpenCL Flags**: We have explored eight standard flags that enable or disable some OpenCL compiler optimizations. For completeness we list here the set of OpenCL flags used, see [6] for explanation: cl-single-precision-constant, cl-denorms-are-zero, cl-opt-disable, cl-mad-enable, cl-no-signed-zeros, cl-finite-math-only, cl-unsafe-math-optimizations, and cl-fast-relaxed-math.
- **LLVM flags**: We have explored five standard flags that enable or disable some LLVM compiler optimizations. They are `-O1`, `-O2`, `-O3`, `-slp-vectorizer`, and `-vectorize-slp-aggressive`.
- **Local work group size**: In OpenCL this refers to the total number of parallel threads running on a compute unit.

- **Vectorization**: Loop vectorization with various vector widths and directions on kernels that allow this optimization.

- **Coarsening degree**: Thread coarsening is an advanced compiler optimization [29] which merges together multiple parallel threads, reducing the total number of threads instantiated. The factor parameter specifies how many threads have been merged. The stride parameter affects the threads' mapping distribution enabling coalesced access patterns. The dimension parameter specifies the dimension affected by the merge.

The compiler parameters affect both power and performance metrics. In addition, accuracy may be affected by some OpenCL flags which cause relaxed maths to be used, for example cl-fast-relaxed-math.

Architecture Space.

The architectural parameters exposed by each platform differ quite significantly. We considered two platforms the ASUS T200TA and the ODROID-XU3. In the case of the ASUS T200TA we are currently only able to select the CPU frequency governor, which in turn scales the CPU frequency and voltage. In this case we have access to only two governors: 'powersave' and 'performance', which set the CPU frequency to the lowest and highest available settings, respectively.

In the case of the ODROID-XU3 platform, we have access to the parameters listed in Table 1:

- **GPU processor frequency**: By default the GPU dynamic voltage and frequency scaling (DVFS) is active and the GPU dynamically adjusts to a particular frequency depending on the performance/power profile of the application. We disable the DVFS and set the GPU frequency to a specific value.

- **Number of active cores**: The number of CPU cores that are active and running, these include the eight "big" (Cortex-A15) and "LITTLE" (Cortex-A7) cores (Section 5.1). By default all 8 cores are active, and we selectively switch off a number of cores.

CPU DVFS is not available on this platform and, therefore, this dimension in the architecture space cannot be explored. The architectural parameters affect both the performance and the power metrics, but not the accuracy. Future approximate computing techniques which for example involve reducing the voltage of compute units, in order to trade off power against the chance of calculation errors, would produce a situation where architectural exploration would involve optimizing across all three targets (rather than just power and runtime).

3.3 Multi-Objective Optimization Goal

Figure 4 presents a fictitious example depicting samples (in green) over a 2-dimensional optimization space [1]. In order to meet the runtime and accuracy thresholds (in dashed lines), the solutions of our exploration are confined to the bottom left region of the space, the targeted prediction area (in black). In a multi-objective optimization, a single solution that minimizes all performance metrics simultaneously does not exist in general. Therefore, attention is paid to Pareto optimal solutions (in blue); which is, solutions that cannot be improved in any of the objectives without degrading at least one of the other objectives. We aim to find the configurations that are simultaneously in the targeted prediction area and on the Pareto front.

[1]For visualization purposes we are only showing two performance metrics, namely the error and the runtime.

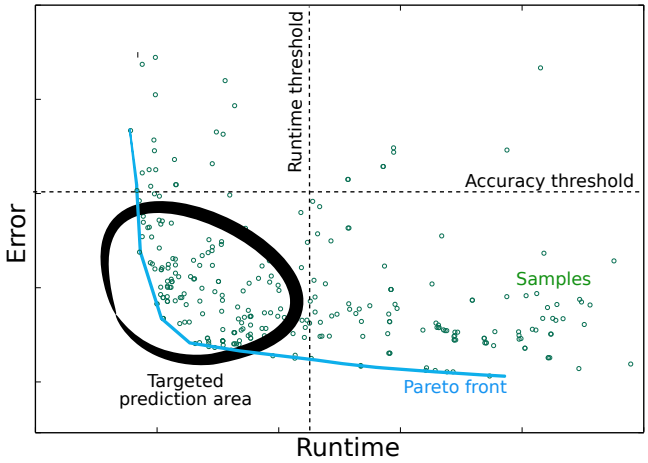


Figure 4: Illustrative example based on fictitious data. This is a two-objective optimization goal in the error and runtime performance metrics. The samples in green are spread all over the space. We are interested in the region highlighted by the black circle, namely the targeted prediction area. The Pareto front is represented in blue.

3.4 Incremental Co-Design Space Exploration

We tackle the co-design space exploration incrementally. We first apply the active learning regressor to the algorithmic parameters. The compiler transformations/optimizations are then applied to the Pareto optimal front points obtained. Since a general tool to drive the compiler exploration is not available for the set of vanilla and advanced compilation parameters that we aim to explore, the compiler space is a mixture of manual and exhaustive search. These are the best performing points of the algorithmic space and are used as an input to the compiler space. The architecture space is then exhaustively evaluated since the size of this space is relatively small (160 points). Our incremental approach enables us to refine the optimal solutions in different steps.

4. SMART SEARCH

The algorithmic parameter space we are investigating is too large to be exhaustively evaluated on the hardware platform. Thus we take the cheaper route of training a predictive machine learning model over a handful of examples (points in the parameter space) evaluated on hardware. We want to use this model to accurately predict the performance over the entire parameter space, while being many orders of magnitude faster as compared to running the application on hardware over a video sequence for millions of parameter settings. Unfortunately since we do not know the performance over the parameter space, we are also unaware of the points for which running a physical experiment will be most informative, in the sense of yielding the greatest increase in

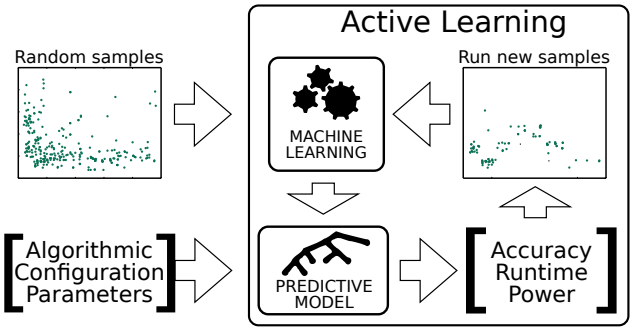


Figure 5: The learning step is based on a tiny subset of the overall algorithmic space; these are the samples that are actually run. Subsequently, the predictive model can predict accuracy, power consumption, and performance of an unseen configuration depending on its parameters.

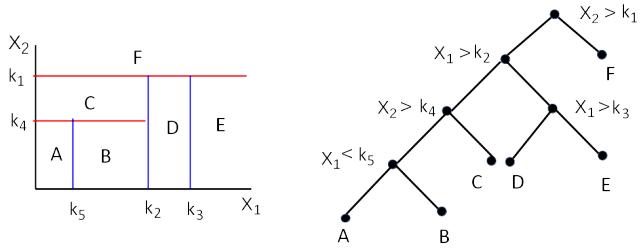


Figure 6: On the left, a 2-dimensional input space with recursive 1-dimensional decision stumps is shown. On the right, a classification decision tree learned over the training data is visualized. Each node represents a portion of the input space.

the prediction accuracy of our model - a classic chicken and egg problem. Thus, we resort to bootstrapping predictive models (three separate randomized decision forests for accuracy, runtime, and power prediction) from a small number of randomly drawn samples in the parameter space. These models are then refined in subsequent iterations by drawing more samples from the parameter space (and retraining over the collective set); the new samples are now drawn to implicitly maximize the prediction accuracy near the respective Pareto optimal fronts. This strategy of letting the predictive model decide which samples will be most beneficial in increasing predictive accuracy over unseen regions of the parameter space is called active learning [16, 41]. Note that we explored a number of base predictive models including artificial neural networks, support vector machines, and nearest neighbors. Our experiments indicated that randomized decision forests outperform these methods, thus we stick to this class of models throughout this paper.

This methodology is depicted in Figure 5 and explained in the next sections.

4.1 Randomized Decision Forest

A decision tree is a tool widely used to formalize decision making processes across a variety of fields. A randomized decision tree is an analogous machine learning model, which "learns" how to classify (or regress) data points based on randomly selected attributes of a set of training examples. The combination of many weak regressors (binary decisions) allows approximating highly non-linear and multimodal functions with great accuracy. Randomized decision forest [12] combines many such decorrelated trees, based on the randomization at the level of training data points and attributes, to yield an even more effective supervised classification and regression model. A decision tree represents a recursive binary partitioning of the input space, and uses a simple decision (a one-dimensional decision threshold) at each non-leaf node that aims at maximizing an "information gain" function. Prediction is performed by "dropping" down the test data point from the root, and letting it traverse a path decided by the node decisions, until it reaches a leaf node. Each leaf node has a corresponding function value (or probability distribution on function values), adjusted according to training data, which is predicted as the function value for the test input. During training, randomization is injected into the procedure to reduce variance and avoid overfitting. This is achieved by training each individual tree on randomly selected subsets of the training samples (also called bagging), as well as by randomly selecting the deciding input variable for each tree node to decorrelate the trees. Figure 6 depicts a decision tree that performs classification over two input dimensions X_1 and X_2, and predicts a class from respective regions.

A regression random forest is built from a set of such decision trees where the leaf nodes output the average of the training data labels, and the output of the whole forest is the average of the predicted results from the different trees. In our experimental setting, we train separate regressors to learn the mapping from our input (parameter) space to each output variable, i.e. the three performance metrics.

4.2 Active Learning

Active learning is a paradigm in supervised machine learning which uses fewer training examples to achieve better prediction accuracy - by iteratively training a predictor, and using the predictor in each iteration to choose the training examples which will increase its accuracy the most. Thus the accuracy of the predictive model is incrementally improved by interleaving exploration and exploitation steps, as shown by the feedback loop in Figure 5. We initialize our base predictors (randomized decision forests) from a very small number of randomly sampled points in the parameter space. For these points the application is evaluated over a video sequence on the hardware platform, yielding accuracy, runtime, and power consumption corresponding to these points (labels in a supervised setting). Since our objective is to accurately estimate the points near the pareto optimal front, we use the current predictor to provide performance values over the entire parameter space and thus estimate the pareto fronts for accuracy, runtime, and power (separately) such as the one in Figure 4. For the next iteration, only parameter points near the predicted pareto front are sampled (and evaluated on hardware), and subsequently used to train new predictors using the entire collection of training points from current and all previous iterations. This process is repeated over a number of iterations. Our experiments (Sect. 5) indicate that this smarter way of searching for highly informative parameter points in fact yields superior predictors as compared to a baseline that uses randomly sampled points alone. Thus by iterating this process several times in the active learning loop, we are able to discover high-quality de-

Machine type	ODROID-XU3	ASUS T200TA
CPU	Samsung Exynos 5422	Intel Atom Z3795
CPU cores	4 (A15) + 4 (A7)	4 Intel Silvermont
CPU GHz	2.0 (A15) 1.4 (A7)	1.6 (2.4 Boost)
GPU	ARM Mali-T628-MP6	Intel HD Graphics
GPU architecture	Midgard 2nd gen.	Bay Trail-T
GPU MHz	600	311 (778 Boost)
OpenCL version	1.1	1.2
Toolkit version	Mali SDK1.1	Beignet v1.1.1
Compiler	gcc 4.8.2	gcc 5.3.1
OS (kernel)	Ubuntu 14.04 (3.10.53)	Fedora 23 (4.2.3)

Table 2: Specification of the platforms selected for experimentation.

sign configurations that lead to good performance outcomes. This approach enables predicting performance metrics over a parameter space comprising of approximately 1,800,000 points in a matter of seconds.

5. EXPERIMENTAL EVALUATION

In this section we describe how we evaluated our novel co-design space exploration techniques. We begin by providing a more detailed description of the target platforms (Section 5.1). We then briefly summarize our key results (Section 5.2), before providing more detail on the results of each stage of our co-design space exploration in Sections 5.3, 5.4, and 5.5.

5.1 Platforms

We use the popular Hardkernel ODROID-XU3 platform, based on the Samsung Exynos 5422, for all of our experiments (refer to Table 2). This board has been previously evaluated for use in UAV applications [34, 24], and is also used in the evaluation of SLAMench [31]. We also considered the ASUS T200TA (refer to Table 2) for comparison during the algorithmic and compilation space. As mentioned earlier, this platform does not provide enough flexibility for a full exploration including the hardware space.

The Exynos 5422 includes a Mali-T628-MP6 GPU alongside ARM's big.LITTLE heterogeneous multiprocessing solution, consisting of four Cortex-A15 "big" performance tuned out-of-order processors, and four Cortex-A7 "LITTLE" energy tuned in-order processors. The Mali-T628-MP6 GPU consists of two separate OpenCL devices: one with four cores and another with two. In our experiments we only use the 4-core OpenCL which excludes partitioning tasks across multiple GPU devices. This is a potential avenue to explore in order to deliver even higher performance within a power budget. The ODROID-XU3 platform has integrated power monitors with on-board voltage/current sensors and split power rails. This allows independent power measurements for the "big" cores, "LITTLE" cores, GPU, and DRAM. The SLAMBench benchmarking framework provides natively an interface to access and log these power sensors.

We also measure performance on an Intel Atom [3] platform in the form of an ASUS Transformer T200 tablet. This contains an Intel Atom Z3795 SoC, which includes a quad-core Intel Atom CPU running at up to 2.4 GHz. An Intel HD Graphics GPU is also present, containing 6 execution units and running at up to 778 MHz. We use the open source Beignet [1] OpenCL runtime which supports version 1.2 of the OpenCL standard and was produced by Intel's Open Technology Center.

Constraint	Speed (FPS)	Max ATE (cm)	Power (Watts)
Default	6.03	4.41	2.77
Best runtime	39.85	4.47	1.47
Best accuracy	1.51	3.30	2.38
Best power	11.92	4.45	0.65
Power $<$ 1W	29.09	4.47	0.98
Power $<$ 2W	39.85	4.47	1.47
FPS $>$ 10	11.92	4.45	0.65
FPS $>$ 20	28.87	4.47	0.91
FPS $>$ 30	32.38	4.47	1.01

Table 3: Best performance on the ODROID-XU3 platform, running KFusion under given constraints.

5.2 Overall Results

We observe that the default configuration provides a frame-rate of 6 FPS for a power budget of 2.77 Watts. Our co-design space exploration results (refer to Table 3) show significantly better frame-rates with reduced power consumption and comparable accuracy. As an example consider a power budget of 1W. Our results show that a configuration exists in the real-time range (29.09 FPS) and with a similar accuracy ATE compared to the default configuration (4.47 cm). The selected best configurations perform well across datasets and in live mode using an actual RGB-D ASUS Xtion Pro camera.

Active learning effectively and consistently pushes the Pareto front towards better solutions. Taking into account the domain layer of the stack unleashes unprecedented performance trade-offs compared to the more usual compiler optimizations. In fact our algorithmic design space exploration provides the greatest improvement on the performance metrics by a large factor (refer to Section 5.3). However, exploration on the hardware parameters shows that important speed/power trade-offs can be obtained in this space. In particular, as we shall see, the greatest improvement in power consumption is provided by exploration of hardware parameters.

5.3 Algorithmic Design Space Exploration

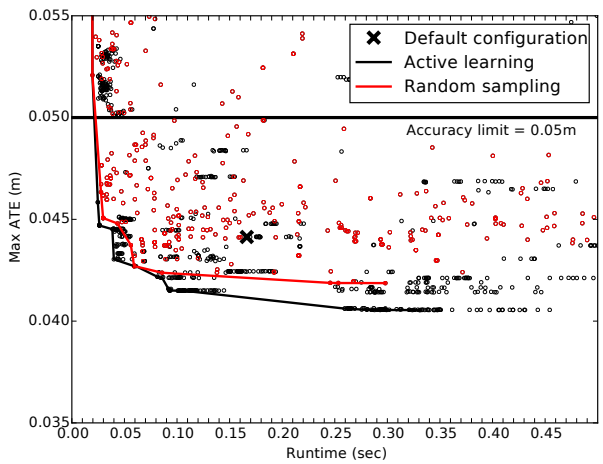
The algorithmic space consists of application parameters summarized in Table 1 and described at the top of Section 3.2. As described in Section 4, we first sample this space at random, and then use active learning in order to push the Pareto front toward better solutions (refer to Figure 5.3).

Sampling.

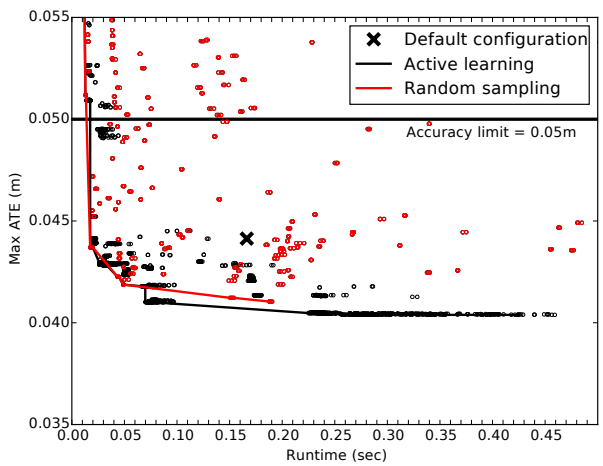
We draw 3,000 uniformly distributed random samples from the parameter space and evaluate the KFusion pipeline on the video stream; for both platforms the cumulated runtimes take roughly 5 days. By using random sampling, we observe that the Pareto front cannot be improved beyond 2,000 samples. Thus, there is an inflection point beyond which random sampling is unproductive.

Active learning.

In order to further explore optimal points in the design space, we employ active learning in conjunction with random decision forest (Section 4.2). For the ODROID-XU3 this produces 1,142 new samples after 6 iterations, thus increasing the total number of samples to 4,142. Note that the



(a) ODROID-XU3



(b) ASUS T200TA

Figure 7: Random sampling (red) and active learning (black). For the sake of visualization we only show two dimensions (max ATE and runtime) of a three dimension plot.

number of samples produced per iteration is not constant as it depends on the predicted points' proximity to the Pareto front. We observe that the number of samples per iteration varies between 100 and 300. The runtime of these new configurations was faster, close to a day, as most of these configurations were good configurations (accurate and fast). The training of the random forest model was fast as well, less than two minutes for every iteration. With the ASUS T200TA platform, 1392 new points has been produced by active learning.

Effectiveness of the active learning method.

Figure 5.3 shows the overall improvement of the Pareto front obtained with active learning (in red) compared to the Pareto obtained with random sampling (in black). For the ODROID-XU3 we observe that random sampling provides a set of 333 valid configurations, i.e. 333 configurations with a max ATE smaller than 5 cm. For the ASUS T200TA, we found 291 valid configurations during the sampling. Furthermore, by using the active learning technique, we observe 642 new possible configurations with an ATE of less than 5 cm on the ODROID-XU3, and 665 on the ASUS T200TA. This means we have produced twice as many valid points as random sampling, for roughly a third of the number of samples. These ratios are an indicator of the effectiveness of our active learning-based prediction model. There is a discrepancy between predicted and measured performance. This is shown by the active learning points in Figure that do not lie on the Pareto front. A performance comparison is also available on Table 4. Note that there are 36 points on the Pareto front for the ODROID-XU3 and 167 points for the ASUS T200TA; these are the configurations forwarded to explore the compiler and architecture parameters in the incremental exploration of section 3.4.

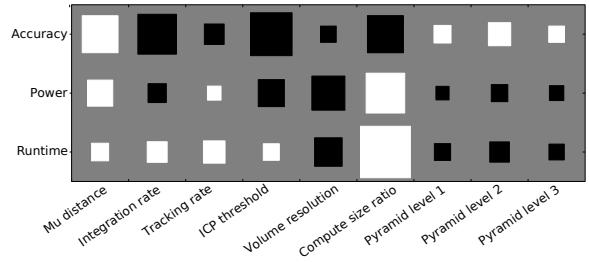


Figure 8: Impact of algorithmic parameters (x-axis) on the performance metrics (y-axis) for the ODROID-XU3 platform (the equivalent diagram for the ASUS T200TA is similar). Bigger squares indicate a higher correlation. A white square denotes a parameter which when increased improves the corresponding metric, whilst a black square shows a worsening.

Relationship between parameters and metrics.

It is particularly interesting to analyze the impact of each algorithmic parameter in isolation. To study the linear relationship of the algorithmic parameters with the performance metrics (frame-rate, accuracy and power) we use the Hinton diagram in Figure 8. In this figure, each square denotes the correlation between a pair (parameter, metric), i.e. the linear relation between a parameter on the x-axis and the performance metric on the y-axis. Note that a bigger square denotes a stronger correlation and vice versa. Furthermore, the square color denotes if a parameter has a positive (white) or negative (black) correlation with the performance metric. For example, an increased *compute size ratio* improves power efficiency and frame-rate but degrades accuracy. This analysis provides a real applicability beyond our design space exploration. It gives an insight into linear relationships between algorithmic parameters and performance goals (frame-rate, accuracy, and power). As shown in the figure there are several parameters that have a non-linear relationship with the performance parameters. It would not be possible for a domain expert to understand these high-dimensional non-linearities, thus emphasizing the importance of automated analysis. For example, *mu*, *integration rate*, *icp threshold*, and the *compute size ratio* are the parameters with strong linear relationship in terms of accuracy, while only *compute size ratio* is strongly linear in execution time.

Constraint		FPS	Error	Power	Volume Resolution	Compute Size Ratio	Pyramid	Integ. Rate	ICP Threshold	μ	Tracking Rate
Default [5]		5.46	4.41	2.1	256	1	10x5x4	2	0.0	0.1	1
Default (measured)		6.03	4.41	2.77	256	1	10x5x4	2	0.0	0.1	1
Best speed	RS	36.11	4.63	2.31	128	4	7x11x7	10	1.0	0.2	1
	AL	**38.28**	**4.47**	**2.16**	128	4	9x3x9	30	1.0	0.2	1
Best accuracy	RS	**3.35**	4.19	2.83	128	1	5x5x11	20	0.0	0.2	1
	AL	3.02	**4.05**	**2.82**	128	1	11x5x5	1	0.0	0.2	1
Best power	RS	19.12	4.73	2.13	128	4	3x11x9	20	0.0	0.1	1
	AL	**38.07**	**4.47**	**2.12**	128	4	9x3x7	30	1.0	0.2	1

Table 4: Comparison of random sampling (RS) best solutions with active learning (AL) search over the algorithmic space for the ODROID-XU3 platform. We list the results for the default configuration given in the original SLAMBench paper [5], and our own results running the same configuration using a newer version of the SLAMBench package. For consistency, in this work our baseline version for performance comparison is the default measured version. The SLAMBench paper provides error as mean ATE over the whole workload, we provide it as the maximum ATE for any frame of the workload.

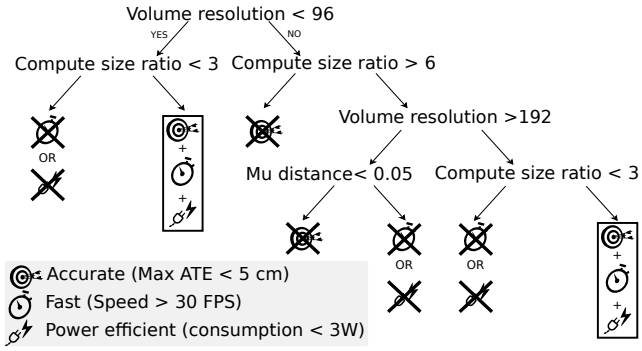


Figure 9: Decision tree showing how algorithmic parameters affect the performance metrics for the ODROID-XU3 platform.

Interpreting the results.

We rank the different algorithmic parameters as a function of their influence on the performance metrics. This is shown as a decision tree (Figure 9) for the ODROID-XU3 platform. The salient advantage of the decision tree is that it can be readily understood. For the sake of presentation we only plot a few levels of the tree.

We observe in our results that the *Volume resolution* is the algorithmic parameter that has the most significant impact on performance. Hence, it is at the root node with a decision threshold of 96. Note that for the *Volume resolution*, 96 is not a valid value but it can be seen as an intermediate of two valid values, i.e. 64 and 128. In addition, note that in Figure 8, the correlation between *Volume resolution* and the performance metrics is relatively small; this further highlights the highly non-linear nature of this parameter. The symbols represent a target performance goal achieved or not achieved, respectively without and with a cross. As we can see, there are two branches that contain configuration points satisfying the three performance metric thresholds depicted in the legend. These branches are *Volume resolution* < 96 and *Compute size ratio* ≥ 3, or *Volume resolution* ≤ 192 and $3 \leq$ *Compute size ratio* ≥ 6. When in a branch we have a performance metric with a cross, that means that there are no configurations in all that sub-tree able to meet that performance metric requirement.

By using the described techniques to explore the algorithmic space, we have obtained a 6.35x improvement in execution time (best speed), and a 23.5% reduction in power consumption (best power), compared to the default configuration on the ODROID-XU3 board. This means that, even without performing further exploration of the compiler and architectural spaces, we are already able to meet our design requirements. However, as will be seen, some runtime improvements and significant reductions in power consumption can still be obtained.

5.4 Compiler Space

Table 1 summarizes the compiler parameters that are explored in our study. For this study we use the 36 Pareto optimal points of the algorithmic space to conduct a compiler design space study. In other words, we take the best performing configurations of the algorithmic space and use this sub-set of optimal points to further explore the compiler space, as explained in section 3.4

Optimizations.

We consider compiler optimizations that only affect the kernels in isolation. Note that kernel transformations such as vectorization, thread coarsening, and OpenCL local workgroup sizes fall in this category. We optimize each kernel independently, which enables us to undertake an exhaustive exploration of the space of our selected optimizations for each kernel. For each vectorizable kernel, we look for every possible loop length vectorization (4 possibilities per axis for a total of 8) and select the best performing configuration. Furthermore, for each kernel we explore 72 different thread coarsening values. Note that a thread coarsening value is a combination of factor, stride, and dimension (see Table 1). To perform this transformation, we use an automatic source to source thread coarsening generation tool [29]. In addition, we also include several LLVM optimizations using the LLVM flags listed in Table 1. In Figure 10, as these optimizations mainly affect the runtime, we only show the impact of exploring the compiler design space on the runtime performance of each kernel of KFusion. These kernels correspond to those summarized in Figure 2 and numerical suffix values differentiate input datasize.

We observe an average runtime speed-up of 23% with the ODROID-XU3, and 16% with the ASUS T200 and a maximum speed-up of 80% for the 'halfSampleRobustIm-

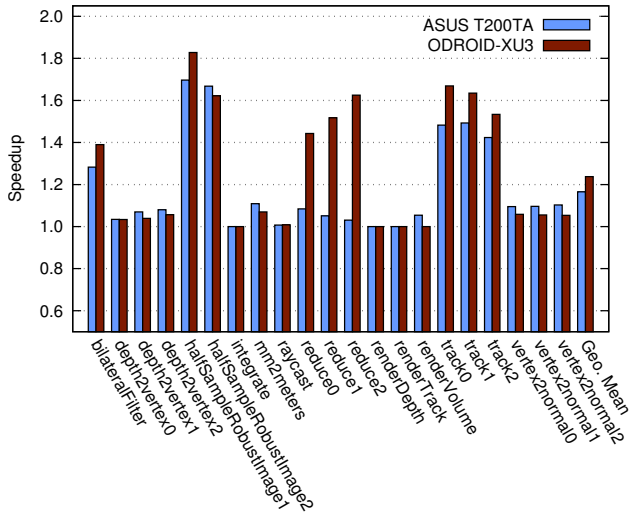


Figure 10: For each Pareto optimal solution in the algorithmic space, we apply some common and advanced compiler optimization techniques. The graph shows the average speed-up obtained for each kernel, for both the ODROID and ASUS platforms.

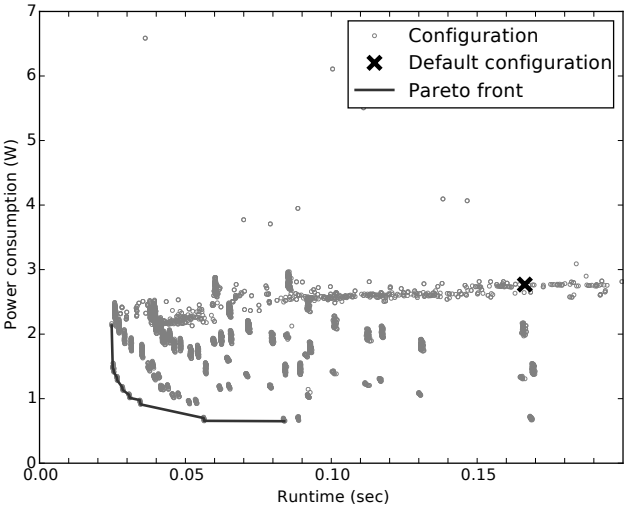


Figure 11: Exhaustive exploration of the architecture design space (grey dots) for the ODROID-XU3 platform. The default configuration (black cross) and the Pareto optimal front (black line) are also depicted.

ageKernel1' kernel on ODROID-XU3. Other kernels achieving speedups are the 'bilateralFilter' (which performs image smoothing in the KFusion front end), and kernels associated with the reduction ('reduce') and tracking ('track') portions of the algorithm. However, we observe that these optimizations are effective only on less computationally intensive kernels, and hence they have little significant impact on the overall execution time of KFusion.

Impact of the compilation parameters.

We observe that the compiler parameters explored in our study provide only modest performance improvements. Specifically, we realize that real-time frame rate, i.e. 30 FPS, cannot be obtained by only applying the compiler optimizations. This highlights the need of co-exploring the algorithmic, compiler, and architecture design space.

When running KFusion on the ODROID-XU3 with the optimized kernels, we observe an average of 6% performance improvement and a maximum of 20% performance improvement over the set of Pareto optimal points obtained from the algorithmic space exploration.

5.5 Architecture Space

The last stage in our incremental co-design space exploration is exploring the architecture parameters in Table 1. Note that the architecture parameters only have an impact on the runtime and power, the ATE is not affected by this space exploration. This exploration will be performed across the Pareto optimal points obtained from the compiler design space stage under the constraint that they are accurate enough, i.e. ATE $<$ 5 cm. This constraint is a reasonable assumption in most applications in the SLAM domain.

Exhaustive exploration.

Since the hardware design space size is only 160 configurations on the ODROID-XU3 platform, it can be exhaustively explored. We visualize the power consumption and runtime dimensions in Figure 11. The black cross depicts the default configuration and the black line is the Pareto front. We observe that a configuration exists that provides a frame-rate of 32.38 FPS (runtime 0.03 seconds) while drawing only 1.01 Watts of power. Thus this represents an interesting configuration that supports real-time performance and, at the same time, consuming minimal power. We further observe that there exists a configuration in the Pareto front that provides a frame-rate of nearly 40 FPS while consuming less than 2 Watts of power. There is also an extreme low-power configuration (0.65W) where we are trading power consumption for a lower frame rate. The improvements obtained from the architecture and compilation spaces can be seen in Table 5.

Constraint		FPS	Error	Power
Best speed	before	38.28	**4.47**	2.16
	after	**39.85**	**4.47**	**1.47**
Best power	before	**38.07**	4.47	2.12
	after	11.92	**4.45**	**0.65**

Table 5: Compiler and architecture space exploration improvements. Our technique has been able to obtain significant improvements in power consumption, without compromising the execution time, and has also found configurations suitable for extremely power constrained environments.

5.6 Discussion

The majority of improvement in runtime came from optimizing at the algorithmic stage. By tuning the various parameters in the co-design space, we were able to achieve significant improvements in both execution time and power consumption (see Table 3). Our optimizations over the compilation and architectural spaces provided minimal improve-

ments in runtime performance, meaning that if we had focused on only the 'lower' two layers of our design space, we would not have been able to reach our design goals with our selected platform. This shows the importance of incorporating domain knowledge into the design space. Although this multi-layered approach may not have converged on an optimal set of solutions, it reduced the size of the design space significantly and allowed us to find good configurations much more quickly.

Importantly, modifying the algorithmic parameters significantly affects the runtime profile of KFusion. As the Hinton plot in Figure 8 shows, each parameter can have varying effects on runtime, accuracy, and power consumption. If we had focused on only the lower tiers of the optimization space, we would have missed this significant opportunity for improvements in runtime and power consumption, and instead obtained only minor improvements.

6. RELATED WORK

The computer vision community primarily focuses on developing accurate algorithms [21, 39], almost always running on high-performance and power hungry systems. As computer vision technology becomes mature, a few benchmarks [40, 15, 38] have attempted to refocus research on runtime constrained contexts. Similarly, new challenges such as the Low-Power Image Recognition Challenge (LPIRC 2016) are emphasizing the importance of low-power embedded implementations of computer vision applications. In this context, recently SLAMBench [31] enabled quantitative, comparable, and validatable experimental research in the form of a benchmark framework for dense 3D scene understanding on a wide range of devices. Adding energy consumption as a metric when evaluating computer vision applications, has enabled energy constrained systems such as battery-powered robots and embedded devices to become evaluation platforms. Zeeshan et al. [42] is a first attempt at exploring SLAM configuration parameters trading off performance for accuracy on embedded systems.

During the last two decades, several design space exploration techniques and frameworks have been used in a variety of different contexts ranging from embedded devices, to compiler research, and system integration [32, 10]. Kang et al. [26] proposed a system which reduces the size of the design space by considering sets of design points to be equivalent. Hu et al. [23] present a user-guided design space exploration framework, allowing the user to identify both good (and bad) design regions, and hence guide the subsequent search. Ansel et al. [8] introduced an extensible and portable framework for empirical performance tuning. It runs an ensemble of search techniques systematically allocating larger budgets to those who perform well, using a multi-armed bandit optimal budget allocation strategy. Norbert et al. tackle the software configurability problem for binary [37] and for both binary and numeric options [36] using a performance-influence model which is based on linear regression. They optimize for execution time on several examples exploring algorithmic and compiler spaces in isolation.

In particular, machine learning (ML) techniques have been recently employed in both architectural and compiler research. Khan et al. [27] employed predictive modeling for cross-program design space exploration in multi-core systems. The techniques developed managed to explore a large design space of chip-multiprocessors running parallel applications with low prediction error. Similarly, Ipek et al. [25] employed an artificial neural network to predict the impact on the performance of hardware parameters, e.g. cache sizes, buffer sizes, of a particular architecture. Furthermore, Lee et al. [28] used polynomial regression to predict power and performance on a multiprocessor design space. Chen et al. [14] suggest that a ML model can be used to produce a relative ranking of design points, rather than predicting their performance precisely.

Regarding compiler optimization research, several efforts to apply ML in this field have been undertaken during the last decade. Cavazos et al. [13] used ML to discover which sequence of compiler optimizations apply better to executed programs. Moreover, the research conducted in [7, 19] employ ML techniques for various compiler optimizations, e.g. loop unrolling, common subexpression elimination, loop hoisting, based on program features.

In contrast to the aforementioned research, to the best of our knowledge, our work is the first to conduct a vertical co-design space exploration, taking into account algorithmic, compiler, and hardware layers in order to solve a three-objective optimization problem. Furthermore, to the best of our knowledge, we show that random forest in conjunction with active learning is effective to focus the search for Pareto optimal configurations in this context.

7. CONCLUSIONS AND FUTURE WORK

We have considered an incremental co-design space exploration on a three-objective optimization goal, optimizing jointly on the runtime, power, and accuracy dimensions. Our incremental co-design is able to combine trade-offs at different levels in the system, refining the Pareto front in subsequent optimization stages. We have been demonstrating our methodology on a popular multi-kernel dense SLAM implementation. As a result, for the first time, this implementation runs in the real-time range on a device with a power budget of 1W. This is a 4.8x improvement in runtime and a 2.8x improvement in power consumption over an hand-tuned implementation by a SLAM domain expert on the same platform for a similar accuracy. This work goes beyond conventional benchmarking in computer systems research by exposing the algorithmic-level design space.

In further work, we will explore how our approach generalizes to different applications, compilers and platforms. We will investigate variable selection methods that reduce the dimension of the space by creating a new feature space and by doing so will enable us to consider larger spaces for bigger mapping problems. There are also a large number of opportunities in transfer learning approaches. In particular, each configuration is likely to give a similar accuracy across a range of devices, and this knowledge might be used to guide exploration toward more interesting points from a power/runtime perspective. Alternatively, we might keep a fixed hardware, and use learned knowledge of the architectural space to more effectively search through design points for different applications running on the same hardware.

8. ACKNOWLEDGMENTS

We acknowledge funding by the EPSRC grant PAMELA EP/K008730/1. M. Luján is funded by a Royal Society University Research Fellowship. We thank the PAMELA Steering Group for the useful discussions.

9. REFERENCES

[1] Beignet. https://www.freedesktop.org/wiki/Software/Beignet/.

[2] Dyson 360 Eye web site. https://www.dyson360eye.com.

[3] Intel Atom Z3795. http://ark.intel.com/products/80267/Intel-Atom-Processor-Z3795-2M-Cache-up-to-2_39-GHz.

[4] Project Tango web site. https://www.google.com/atap/projecttango.

[5] SLAMBench web site. http://apt.cs.manchester.ac.uk/projects/PAMELA/tools/SLAMBench.

[6] *OpenCL 1.1 Specification*, Sept. 2010.

[7] F. Agakov, E. Bonilla, J. Cavazos, B. Franke, G. Fursin, M. F. P. O'Boyle, J. Thomson, M. Toussaint, and C. K. I. Williams. Using machine learning to focus iterative optimization. In *Proceedings of the International Symposium on Code Generation and Optimization*, CGO '06, pages 295–305, Washington, DC, USA, 2006. IEEE Computer Society.

[8] J. Ansel, S. Kamil, K. Veeramachaneni, J. Ragan-Kelley, J. Bosboom, U.-M. O'Reilly, and S. Amarasinghe. Opentuner: an extensible framework for program autotuning. In *Proceedings of the 23rd international conference on Parallel architectures and compilation*, pages 303–316. ACM, 2014.

[9] ARM Ltd. big.little technology.

[10] F. Balarin, Y. Watanabe, H. Hsieh, L. Lavagno, C. Passerone, and A. Sangiovanni-Vincentelli. Metropolis: an integrated electronic system design environment. In *IEEE Computer*, volume 36, pages 45–52, April 2003.

[11] P. J. Besl and N. D. McKay. Method for registration of 3-D shapes. In *Robotics-DL tentative*. Int. Society for Optics and Photonics, 1992.

[12] L. Breiman. *Classification And Regression Trees*. Chapman and Hall, London, UK, 1984.

[13] J. Cavazos, C. Dubach, F. Agakov, E. Bonilla, M. F. P. O'Boyle, G. Fursin, and O. Temam. Automatic performance model construction for the fast software exploration of new hardware designs. In *Proceedings of the 2006 International Conference on Compilers, Architecture and Synthesis for Embedded Systems*, CASES '06, pages 24–34, New York, NY, USA, 2006. ACM.

[14] T. Chen, Q. Guo, K. Tang, O. Temam, Z. Xu, Z.-H. Zhou, and Y. Chen. Archranker: A ranking approach to design space exploration. *SIGARCH Comput. Archit. News*, 42(3):85–96, June 2014.

[15] J. Clemons, H. Zhu, S. Savarese, and T. Austin. MEVBench: A mobile computer vision benchmarking suite. In *IISWC*, 2011.

[16] C. Cohn, L. Atlas, and R. Ladner. Improving generalization with active learning. *Machine Learning*, 15(2):201–221, 1994.

[17] H. Esmaeilzadeh, E. Blem, R. St. Amant, K. Sankaralingam, and D. Burger. Dark silicon and the end of multicore scaling. In *Proceedings of the 38th Annual International Symposium on Computer Architecture*, ISCA '11, pages 365–376, New York, NY, USA, 2011. ACM.

[18] M. Fallon, P. Marion, R. Deits, T. Whelan, M. Antone, J. McDonald, and R. Tedrake. Continuous humanoid locomotion over uneven terrain using stereo fusion. In *ICHR*, 2015.

[19] G. Fursin, Y. Kashnikov, A. Memon, Z. Chamski, O. Temam, M. Namolaru, E. Yom-Tov, B. Mendelson, A. Zaks, E. Courtois, F. Bodin, P. Barnard, E. Ashton, E. Bonilla, J. Thomson, C. K. Williams, and M. O'Boyle. Milepost gcc: Machine learning enabled self-tuning compiler. *International Journal of Parallel Programming*, 39(3):296–327, 2011.

[20] A. Handa, V. Patraucean, V. Badrinarayanan, S. Stent, and R. Cipolla. SceneNet: Understanding Real World Indoor Scenes With Synthetic Data. *ArXiv e-prints 1511.07041*, 2015.

[21] A. Handa, T. Whelan, J. McDonald, and A. Davison. A Benchmark for RGB-D Visual Odometry, 3D Reconstruction and SLAM. In *ICRA*, 2014.

[22] J. L. Henning. SPEC CPU2006 benchmark descriptions. *ACM SIGARCH Computer Architecture News*, 2006.

[23] X. Hu, G. Greenwood, S. Ravichandran, and G. Quan. A framework for user assisted design space exploration. In *Proceedings of 36th Design Automation Conference*, pages 414–419, 1999.

[24] D. Hulens, T. Goedemé, and J. Verbeke. How to choose the best embedded processing platform for on-board UAV image processing? *Proceedings VISAPP 2015*, pages 1–10, 2015.

[25] E. Ïpek, S. A. McKee, R. Caruana, B. R. de Supinski, and M. Schulz. Efficiently exploring architectural design spaces via predictive modeling. *SIGARCH Comput. Archit. News*, 34(5):195–206, Oct. 2006.

[26] E. Kang, E. Jackson, and W. Schulte. An approach for effective design space exploration. In *Foundations of Computer Software. Modeling, Development, and Verification of Adaptive Systems*, volume 6662 of *Lecture Notes in Computer Science*, pages 33–54. Springer Berlin Heidelberg, 2011.

[27] S. Khan, P. Xekalakis, J. Cavazos, and M. Cintra. Using Predictive Modeling for Cross-Program Design Space Exploration in Multicore Systems. In *Proceedings of the 16th International Conference on Parallel Architecture and Compilation Techniques*, PACT '07, pages 327–338, Washington, DC, USA, 2007. IEEE Computer Society.

[28] B. C. Lee and D. M. Brooks. Accurate and efficient regression modeling for microarchitectural performance and power prediction. *SIGARCH Comput. Archit. News*, 34(5):185–194, Oct. 2006.

[29] A. Magni, C. Dubach, and M. F. O'Boyle. A large-scale cross-architecture evaluation of thread-coarsening. In *Proc. of SC13: Int. Conf. for High Performance Computing, Networking, Storage and Analysis*. ACM, 2013.

[30] S. Moll. Decompilation of LLVM IR. Master's thesis, 2011.

[31] L. Nardi, B. Bodin, M. Z. Zia, J. Mawer, A. Nisbet, P. H. J. Kelly, A. J. Davison, M. Luján, M. F. P. O'Boyle, G. Riley, N. Topham, and S. Furber. Introducing SLAMBench, a performance and accuracy benchmarking methodology for SLAM. In *ICRA*, 2015.

[32] S. Neema, J. Sztipanovits, G. Karsai, and K. Butts. Constraint-based design-space exploration and model synthesis. In *Embedded Software*, volume 2855 of *Lecture Notes in Computer Science*, pages 290–305. Springer Berlin Heidelberg, 2003.

[33] R. A. Newcombe, S. Izadi, O. Hilliges, D. Molyneaux, D. Kim, A. J. Davison, P. Kohli, J. Shotton, S. Hodges, and A. Fitzgibbon. KinectFusion: Real-time dense surface mapping and tracking. In *ISMAR*, 2011.

[34] C. Papachristos, D. Tzoumanikas, and A. Tzes. Aerial robotic tracking of a generalized mobile target employing visual and spatio-temporal dynamic subject perception. In *Intelligent Robots and Systems (IROS), 2015 IEEE/RSJ International Conference on*, pages 4319–4324, Sept 2015.

[35] R. Salas-Moreno, B. Glocker, P. H. J. Kelly, and A. J. Davison. Dense planar SLAM. In *ISMAR*, 2014.

[36] N. Siegmund, A. Grebhahn, S. Apel, and C. Kästner. Performance-influence models for highly configurable systems. In *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering*, pages 284–294. ACM, 2015.

[37] N. Siegmund, S. S. Kolesnikov, C. Kästner, S. Apel, D. Batory, M. Rosenmüller, and G. Saake. Predicting performance via automated feature-interaction detection. In *Proceedings of the 34th International Conference on Software Engineering*, pages 167–177. IEEE Press, 2012.

[38] S. Song, S. P. Lichtenberg, and J. Xiao. Sun rgb-d: A rgb-d scene understanding benchmark suite. In *Computer Vision and Pattern Recognition (CVPR), 2015 IEEE Conference on*, pages 567–576, June 2015.

[39] J. Sturm, N. Engelhard, F. Endres, W. Burgard, and D. Cremers. A benchmark for the evaluation of RGB-D SLAM systems. In *IROS*, 2012.

[40] S. K. Venkata, I. Ahn, D. Jeon, A. Gupta, C. Louie, S. Garcia, S. Belongie, and M. B. Taylor. SD-VBS: The San Diego vision benchmark suite. In *IISWC*, 2009.

[41] M. Warmuth, G. Ratsch, M. Mathieson, J. Liao, and C. Lemmon. Active learning in the drug discovery process. In *Neural Information Processing Systems (NIPS)*, 2001.

[42] M. Z. Zia, L. Nardi, A. Jack, E. Vespa, B. Bodin, P. H. J. Kelly, and A. J. Davison. Comparative Design Space Exploration of Dense and Semi-Dense SLAM. In *ICRA*, 2016.