INVITED PAPER

Navigating the Landscape for Real-Time Localization and Mapping for Robotics and Virtual and Augmented Reality

This paper shows for the important example of simultaneous localization and mapping (SLAM) the compilation and tuning techniques necessary to reach high performance.

By Sajad Saeedi, Bruno Bodin, Harry Wagstaff, Andy Nisbet, Luigi Nardi, John Mawer, Nicolas Melot, Oscar Palomar, Emanuele Vespa, Tom Spink, Cosmin Gorgovan, Andrew Webb, James Clarkson, Erik Tomusk, Thomas Debrunner, Kuba Kaszyk, Pablo Gonzalez-De-Aledo, Andrey Rodchenko, Graham Riley, Christos Kotselidis, Björn Franke, Michael F. P. O'Boyle, Andrew J. Davison, Paul H. J. Kelly, Mikel Luján, and Steve Furber

ABSTRACT | Visual understanding of 3-D environments in real time, at low power, is a huge computational challenge. Often referred to as simultaneous localization and mapping (SLAM), it is central to applications spanning domestic and industrial robotics, autonomous vehicles, and virtual and augmented reality. This paper describes the results of a major research effort to assemble the algorithms, architectures, tools, and systems software needed to enable delivery of SLAM, by supporting applications specialists in selecting and configuring the appropriate algorithm and the appropriate hardware, and compilation pathway, to meet their performance, accuracy, and energy consumption goals. The major contributions we present are: 1) tools and methodology for systematic quantitative evaluation of SLAM algorithms; 2) automated, machine-learning-guided exploration of the algorithmic and implementation design space with respect to multiple objectives; 3) end-to-end simulation tools to enable optimization of heterogeneous, accelerated architectures for the specific algorithmic requirements of the various SLAM algorithmic approaches; and 4) tools for delivering, where appropriate, accelerated, adaptive SLAM solutions in a managed, JIT-compiled, adaptive runtime context.

KEYWORDS | Automatic performance tuning; hardware simulation; scheduling; simultaneous localization and mapping (SLAM)

Manuscript received October 3, 2017; revised May 24, 2018; accepted June 29, 2018. Date of publication August 14, 2018; date of current version October 25, 2018. This work was supported by the Engineering and Physical Sciences Research Council (EPSRC) under Grant EP/K008730/1, PAMELA Project. *(Corresponding author: Sajad Saeedi.)*

S. Saeedi, **N. Melot**, **E. Vespa**, **T. Debrunner**, **P. Gonzalez-de-Aledo**, **A. J. Davison**, and **P. H. J. Kelly** are with the Department of Computing, Imperial College London, London SW7 2AZ, U.K. (e-mail: s.saeedi@imperial.ac.uk).

B. Bodin, **H. Wagstaff**, **T. Spink**, **E. Tomusk**, **K. Kaszyk**, **B. Franke**, and **M. F. P. O'Boyle** are with the School of Informatics, University of Edinburgh, Edinburgh EH8 9AB, U.K.

A. Nisbet, J. Mawer, O. Palomar, C. Gorgovan, A. Webb, J. Clarkson, A. Rodchenko, G. Riley, C. Kotselidis, M. Luján, and **S. Furber** are with the School of Computer Science, University of Manchester, Manchester M139PL, U.K.

L. Nardi is with the Division of Electrical Engineering - Computer Systems, Stanford University, Stanford, CA 94305-9025 USA.

Digital Object Identifier 10.1109/JPROC.2018.2856739

I. INTRODUCTION

Programming increasingly heterogeneous systems for emerging application domains is an urgent challenge. One particular domain with massive potential is *real-time 3-D scene understanding*, poised to effect a radical transformation in the engagement between digital devices and the physical human world. In particular, visual

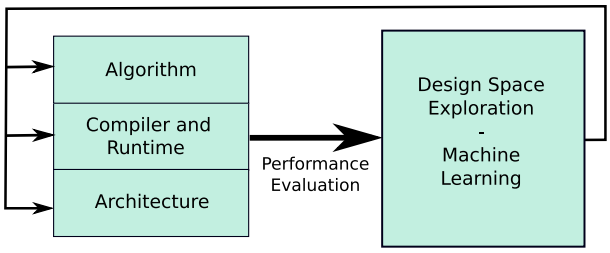

Fig. 1. ***The objective of the paper is to create a pipeline that aligns computer vision requirements with hardware capabilities. The paper's focus is on three layers: algorithms, compiler and runtime, and architecture. The goal is to develop a system that allows us to achieve power and energy efficiency, speed and runtime improvement, and accuracy/robustness at each layer and also holistically through design space exploration and machine learning techniques.***

simultaneous localization and mapping (SLAM), defined as determining the position and orientation of a moving camera in an unknown environment by processing image frames in real time, has emerged to be an enabling technology for robotics and virtual/augmented reality applications.

The objective of this work is to build the tools to enable the computer vision pipeline architecture to be designed so that SLAM requirements are aligned with hardware capability. Since SLAM is computationally very demanding, several subgoals are defined, developing systems with: 1) power and energy efficiency; 2) speed and runtime improvement; and 3) improved results in terms of accuracy and robustness. Fig. 1 presents an overview of the directions explored. At the first stage, we consider different layers of the system including architecture, compiler and runtime, and computer vision algorithms. Several distinct contributions have been presented in these three layers, explained throughout the paper. These contributions include novel benchmarking frameworks for SLAM algorithms, various scheduling techniques for software performance improvement, and "functional" and "detailed" hardware simulation frameworks. Additionally, we present holistic optimization techniques, such as design space exploration (DSE), that allows us to take into account all these layers together and optimize the system holistically to achieve the desired performance metrics.

The major contributions we present are:

- tools and methodology for systematic quantitative evaluation of SLAM algorithms;
- automated, machine-learning-guided exploration of the algorithmic and implementation design space with respect to multiple objectives;
- end-to-end simulation tools to enable optimization of heterogeneous, accelerated architectures for the specific algorithmic requirements of the various SLAM algorithmic approaches;
- tools for delivering, where appropriate, accelerated, adaptive SLAM solutions in a managed, JIT-compiled, adaptive runtime context.

This paper is an overview of a large body of work unified by these common objectives: to apply software synthesis and automatic performance tuning in the context of compilers and library generators; performance engineering; program generation, languages, and hardware synthesis. We specifically target mobile, embedded, and wearable contexts, where trading quality-of-result against energy consumption is of critical importance. The key significance of the work lies, we believe, in showing the importance and the feasibility of extending these ideas across the full stack, incorporating algorithm selection and configuration into the design space along with code generation and hardware levels of the system.

A. Background

Based on the structure shown in Fig. 1, in this section, background material for the following topics is presented very briefly:

- computer vision;
- system software;
- computer architecture;
- model-based design space exploration.

1) Computer Vision: In computer vision and robotics community, SLAM is a well-known problem. Using SLAM, a sensor, such as a camera, is able to localize itself in an unknown environment by incrementally building a map and at the same time localizing itself within the map. Various methods have been proposed to solve the SLAM problem, but robustness and real-time performance is still challenging [1]. From the mid 1990s onwards, a strong return has been made to a model-based paradigm enabled primarily by the adoption of probabilistic algorithms [2] which are able to cope with the uncertainty in all real sensor measurements [3]. A breakthrough was when it was shown to be feasible using computer vision applied to commodity camera hardware. The MonoSLAM system offered real-time 3-D tracking of the position of a hand-held or robot-mounted camera while reconstructing a sparse point cloud of scene landmarks [4]. Increasing computer power has since enabled previously "offline" vision techniques to be brought into the real-time domain; parallel tracking and mapping (PTAM) made use of classical bundle adjustment within a real-time loop [5]. Then live dense reconstruction methods, dense tracking and mapping (DTAM) using a standard single camera [6] and KinectFusion using a Microsoft Kinect depth camera [7] showed that surface reconstruction can be a standard part of a live SLAM pipeline, making use of GPU-accelerated techniques for rapid dense reconstruction and tracking.

KinectFusion is an important research contribution and has been used throughout this paper in several sections, including in SLAMBench benchmarking (Section II-A), in improved mapping and path planning in robotic applications (Section II-B), in Diplomat static scheduling (Section III-A2), in Tornado and MaxineVM dynamic

scheduling (Sections III-B1 and III-B2), in MaxSim hardware profiling (Section IV-B2), and various design space exploration and crowdsourcing methods (Section V).

KinectFusion models the occupied space only and tells nothing about the free space which is vital for robot navigation. In this paper, we present a method to extend KinectFusion to model free space as well (Section II-B). Additionally, we introduce two benchmarking frameworks, SLAMBench and SLAMBench2 (Section II-A). These frameworks allow us to study various SLAM algorithms, including KinectFusion, under different hardware and software configurations. Moreover, a new sensor technology, focal-plane sensor-processor arrays, is used to develop scene understanding algorithms, operating at very high frame rates with very low power consumption (Section II-C).

2) System Software: Smart scheduling strategies can bring significant performance improvement regarding execution time [8] or energy consumption [9]–[11] by breaking an algorithm into smaller units, distributing the units between cores or intellectual properties (IPs) available, and adjusting the voltage and frequency of the cores. Scheduling can be done either statically or dynamically. Static scheduling requires extended knowledge about the application, i.e., how an algorithm can be broken into units, and how these units behave in different settings. Decomposing an algorithm this way impacts a static scheduler's choice in allocating and mapping resources to computation units, and therefore it needs to be optimized. In this paper, two static scheduling techniques are introduced (Section III-A) including idiom-based compilation and Diplomat, a task-graph framework that exploits static dataflow analysis to perform CPU/GPU mapping.

Since static schedulers do not operate online, optimization time is not a primary concern. However, important optimization opportunities may depend on the *data* being processed; therefore, dynamic schedulers have more for obtaining the best performance. In this paper, two novel dynamic scheduling techniques are introduced including MaxineVM, a research platform for managed runtime languages executing on ARMv7, and Tornado, a heterogeneous task-parallel programming framework designed for heterogeneous systems where the specific configurations of CPUs, GPGPUs, FPGAs, DSPs, etc. in a system are not known until runtime (Section III-B).

In contrast, dynamic schedulers cannot spend too much processing power to find good solutions, as the performance penalty may outweigh the benefits they bring. Quasi-static scheduling is a compromising approach that statically computes a good schedule and further improves it online depending on runtime conditions [12]. A hybrid scheduling technique is introduced called power-aware code generation, which is a compiler-based approach to runtime power management for heterogeneous cores (Section III-C).

3) Computer Architecture: It has been shown that moving to a dynamic heterogeneous model, where the use of hardware resources and the capabilities of those resources are adjusted at runtime, allows far more flexible optimization of system performance and efficiency [13], [14]. Simulation methods, such as memory and instruction set simulation, are powerful tools to design and evaluate such systems. A large number of simulation tools are available [15]; in this paper we further improve upon current tools by introducing novel "functional" and "detailed" hardware simulation packages, that can simulate individual cores and also complete CPU/GPU systems (Section IV-A). Also novel profiling (Section IV-B) and specialization (Section IV-C) techniques are introduced which allow us to custom-design chips for SLAM and computer vision applications.

4) Model-Based Design Space Exploration: Machine learning has rapidly emerged as a viable means to automate sequential optimizing compiler construction. Rather than hand-craft a set of optimization heuristics based on compiler expert insight, learning techniques automatically determine how to apply optimizations based on statistical modelling and learning. Its great advantage is that it can adapt to changing platforms as it has no *a priori* assumptions about their behavior. There are many studies showing it outperforms human-based approaches [16]–[19].

Recent work shows that machine learning can automatically port across architecture spaces with no additional learning time, and can find different, appropriate, ways of mapping program parallelism for different parallel platforms [20], [21]. There is now ample evidence from previous research, that design space exploration based on machine learning provides a powerful tool for optimizing the configuration of complex systems both statically and dynamically. It has been used from the perspective of single-core processor design [22], the modelling and prediction of processor performance [23], the dynamic reconfiguration of memory systems for energy efficiency [24], the design of SoC interconnect architectures [25], and power management [24]. The DSE methodology will address this paper's goals from the perspective of future many-core systems, extending beyond compilers and architecture to elements of the system stack including application choices and runtime policies. In this paper, several DSE related works are introduced. Multi-domain DSE performs exploration on hardware, software, and algorithmic choices (Section V-A1). With multidomain DSE, it is possible to compromise between metrics such as runtime speed, power consumption, and SLAM accuracy. In motion-aware DSE (Section V-A2), we develop a comprehensive DSE that also takes into account the complexity of the environment being modelled, including the photometric texture of the environment, the geometric shape of the environment, and the speed of the camera in the environment. DSE works allow us to design applications that can optimally choose a set of hardware, software, and algorithmic parameters

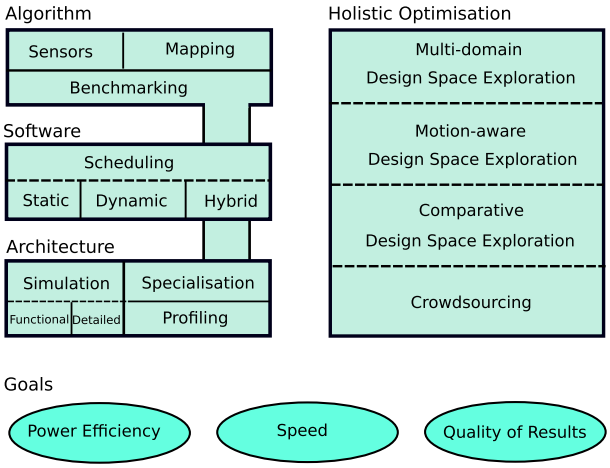

Fig. 2. ***Outline of the paper. The contributions of the paper have been organized under four sections, shown with solid blocks. These blocks cover algorithmic, software, architecture, and holistic optimization domains. Power efficiency, runtime speed, and quality of results are the subgoals of the project. The latter includes metrics such as accuracy of model reconstruction, accuracy of trajectory, and robustness.***

meeting certain desired performance metrics. One example application is active SLAM (Section V-A2a).

B. Outline

Real-time 3-D scene understanding is the main driving force behind this work. Three-dimensional scene understanding has various applications in wearable devices, mobile systems, personal assistant devices, Internet of Things, and many more. Throughout this paper, we aim to answer the following questions: 1) How can we improve 3-D scene understanding (especially SLAM) algorithms? 2) How can we improve power performance for heterogeneous systems? 3) How can we reduce the development complexity of hardware and software? As shown in Fig. 2, we focus on four design domains: computer vision algorithms, software, hardware, and holistic optimization methods. Several novel improvements have been introduced, organized as shown in Fig. 2.

- Section II (Algorithm) explains the algorithmic contributions such as using novel sensors, improving dense mapping, and novel benchmarking methods.
- Section III (Software) introduces software techniques for improving system performance, including various types of scheduling.
- Section IV (Architecture) presents hardware developments, including simulation, specialization, and profiling techniques.
- Section V (Holistic Optimization) introduces holistic optimization approaches, such as design space exploration and crowdsourcing.
- Section VI summarizes the work.

II. COMPUTER VISION ALGORITHMS AND APPLICATIONS

Computer vision algorithms are the main motivation of the paper. We focus mainly on SLAM. Within the past few decades, researchers have developed various SLAM algorithms, but few tools are available to compare and benchmark these algorithms and evaluate their performance on the available diverse hardware platforms. Moreover, the general research direction is also moving towards making the current algorithms more robust to eventually make them available in industries and our everyday life. Additionally, as the sensing technologies progress, the pool of SLAM algorithms becomes more diverse and fundamentally new approaches need to be invented.

This section presents algorithmic contributions from three different aspects. As shown in Fig. 3, three main topics are covered: 1) benchmarking tools to compare the performance of the SLAM algorithms; 2) improved probabilistic mapping; and 3) new sensor technologies for scene understanding.

A. Benchmarking: Evaluation of SLAM Algorithms

Real-time computer vision and SLAM offer great potential for a new level of scene modelling, tracking, and real environmental interaction for many types of robots, but their high computational requirements mean that implementation on mass market embedded platforms is challenging. Meanwhile, trends in low-cost, low-power processing are towards massive parallelism and heterogeneity, making it difficult for robotics and vision researchers to implement their algorithms in a performance-portable way.

To tackle the aforementioned challenges, in this section, two computer vision benchmarking frameworks are introduced: SLAMBench and SLAMBench2. Benchmarking is a scientific method to compare the performance of different hardware and software systems. Both benchmarking frameworks share common functionalities, but their objectives are different. While SLAMBench provides a framework that is able to benchmark various implementations

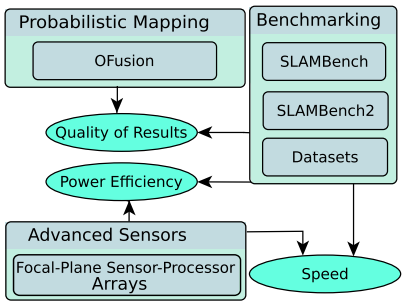

Fig. 3. ***Algorithmic contributions include benchmarking tools, advanced sensors, and improved probabilistic mapping.***

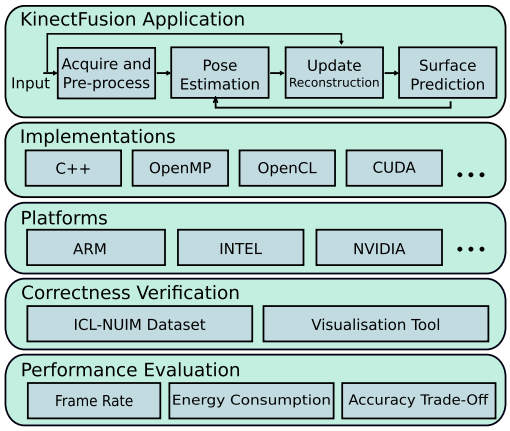

Fig. 4. ***SLAMBench enables benchmarking of the KinectFusion algorithm on various types of platforms by providing different implementations such as C++, OpenMP, CUDA, and OpenCL.***

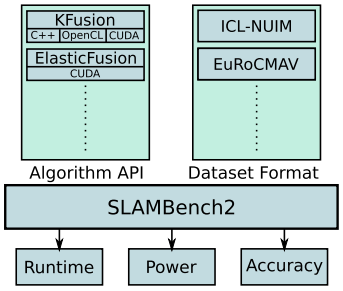

Fig. 5. ***SLAMBench2 allows multiple algorithms (and implementations) to be combined with a wide array of data sets. A simple API and data set make it easy to interface with new algorithms.***

of KinectFusion, SLAMBench2 provides a framework that is able to benchmark various different SLAM algorithms in their original implementations.

Additionally, to systemically choose the proper data sets to evaluate the SLAM algorithms, we introduce a data set complexity scoring method. All these projects allow us to optimize power, speed, and accuracy.

1) SLAMBench: As a first approach to investigate SLAM algorithms, we introduced SLAMBench [26], a publicly available software framework which represents a starting point for quantitative, comparable, and validatable experimental research to investigate tradeoffs in performance, accuracy, and energy consumption of a dense RGB-D SLAM system. SLAMBench provides a KinectFusion [7] implementation, inspired by the open-source KFusion implementation [27]. SLAMBench provides the same KinectFusion in the C++, OpenMP, CUDA, and OpenCL variants, and harnesses the ICL-NUIM synthetic RGB-D data set [28] with trajectory and scene ground truth for reliable accuracy comparison of different implementation and algorithms. The overall vision of the SLAMBench framework is shown in Fig. 4; refer to [26] for more information.

Third parties have provided implementations of SLAMBench in additional emerging languages:

- C++ SyCL for OpenCL Khronos Group standard [29];
- platform-neutral compute intermediate language for accelerator programming PENCIL [30], the PENCIL SLAMBench implementation can be found in [31].

As demonstrated in Fig. 2, SLAMBench has enabled us to do more research in algorithmic, software, and architecture domains, explained throughout the paper. Examples include Diplomat static scheduling (Section III-A2), Tornado dynamic scheduling (Sections III-B1), MaxSim hardware profiling (Section IV-B2), multidomain design space exploration (Section V-A1), comparative design space exploration (Section V-A3), and crowdsourcing (Section V-B).

2) SLAMBench2: SLAMBench has had substantial success within both the compiler and architecture realms of academia and industry. The SLAMBench performance evaluation framework is tailored for the KinectFusion algorithm and the ICL-NUIM input data set. However, in SLAMBench 2.0, we reengineered SLAMBench to have more modularity by integrating two major features [32]. Firstly, a SLAM API has been defined, which provides an easy interface to integrate any new SLAM algorithms into the framework. Secondly, there is now an I/O system in SLAMBench2 which enables the easy integration of new data sets and new sensors (see Fig. 5). Additionally, SLAMBench2 features a new set of algorithms and data sets from among the most popular in the computer vision community, Table 1 summarizes these algorithms.

The works in [40] and [41] present benchmarking results, comparing several SLAM algorithms on various hardware platforms; however, SLAMBench2 provides a framework that researchers can easily integrate and use to explore various SLAM algorithms.

3) Data Sets: Research papers on SLAM often report performance metrics such as pose estimation accuracy, scene reconstruction error, or energy consumption. The reported performance metrics may not be representative of how well an algorithm will work in real-world applications.

Table 1 List of SLAM Algorithms Currently Integrated in SLAMBench2. These Algorithms Provide Either Dense, Semi-Dense, or Sparse Reconstructions [32]

Algorithm	Type	Implementations
ElasticFusion [33]	Dense	CUDA
InfiniTAM [34]	Dense	C++, OpenMP, CUDA
KinectFusion [7]	Dense	C++, OpenMP, OpenCL, CUDA
LSD-SLAM [35]	Semi-Dense	C++, PThread
ORB-SLAM2 [36]	Sparse	C++
MonoSLAM [37]	Sparse	C++, OpenCL
OKVIS [38]	Sparse	C++
PTAM [5]	Sparse	C++
SVO [39]	Sparse	C++

Table 2 Complexity Level Metrics Using Information Divergence [44]

Dataset	Trajectory	Max	Mean	Variance
ICL-NUIM	lr_kt0	0.0250	0.0026	0.0014
	lr_kt1	0.0183	0.0026	0.0012
	lr_kt2	0.0427	0.0032	0.0023
	lr_kt3	0.0352	0.0032	0.0023

Additionally, as the diversity of the data sets is growing, it becomes a challenging issue to decide which and how many data sets should be used to compare the results. To address this concern, not only have we categorized data sets according to their complexity in terms of trajectory and environment, but also we have proposed new synthetic data sets with highly detailed scene and realistic trajectories [42], [43].

In general, data sets do not come with a measure of complexity level, and thus the comparisons may not reveal all strengths or weaknesses of a new SLAM algorithm. In [44], we proposed to use frame-by-frame Kullback-Leibler divergence as a simple and fast metric to measure the complexity of a data set. Across all frames in a data set, mean divergence and the variance of divergence were used to assess the complexity. Table 2 shows some of these statistics for ICL-NUIM sequences for intensity divergence. Based on the reported trajectory error metrics of the ElasticFusion algorithm [33], data sets lr_kt2 and lr_kt3 are more difficult than lr_kt0 and lr_kt1. Using the proposed statistical divergence, these difficult trajectories have a higher complexity score as well.

B. OFusion: Probabilistic Mapping

Modern dense volumetric methods based on signed distance functions such as DTAM [6] or explicit point clouds, such as ElasticFusion [33], are able to recover high quality geometric information in real time. However, they do not explicitly encode information about empty space which essentially becomes equivalent to unmapped space. In various robotic applications this could be a problem as many navigation algorithms require explicit and persistent knowledge about the mapped empty space. Such information is instead well encoded in classic occupancy grids, which, on the other hand, lack the ability to faithfully represent the surface boundaries. Loop *et al*. [45] proposed a novel probabilistic fusion framework aiming at closing such an information gap by employing a continuous occupancy map representation in which the surface boundaries are well defined. Targeting real-time robotics applications, we have extended such a framework to make it suitable for the incremental tracking and mapping typical of an exploratory SLAM system. The new formulation, denoted as OFusion [46], allows robots to seamlessly perform camera tracking, occupancy grid mapping and surface reconstruction at the same time. As shown in Table 3, OFusion not only encodes the free space, but also performs at the same level or better than state-of-the-art volumetric SLAM pipelines such as KinectFusion [7] and InfiniTAM [34] in terms of mean absolute trajectory error (ATE). To demonstrate the effectiveness of our approach we implemented a simple path planning application on top of our mapping pipeline. We used Informed RTT* [47] to generate a collision-free 3-m-long trajectory between two obstructed start-goal endpoints, showing the feasibility to achieve tracking, mapping and planning in a single integrated control loop in real time.

Table 3 Absolute Trajectory Error (ATE), in Meters, Comparison Between KinectFusion (TSDF), Occupancy Mapping (OFusion), and InfiniTAM Across Sequences From the ICL-NUIM and TUM RGB-D Data Sets. Cross Signs Indicate Tracking Failure

Trajectory	TSDF	OFusion	InfiniTAM
ICL-NUIM lr_kt0	0.0113	0.2289	0.3052
ICL-NUIM lr_kt1	0.0117	0.0170	0.0214
ICL-NUIM lr_kt2	0.0040	0.0055	0.1725
ICL-NUIM lr_kt3	0.7582	0.0904	0.4858
TUM fr1_xyz	0.0295	0.0322	0.0273
TUM fr1_floor	×	×	×
TUM fr1_plant	×	×	×
TUM fr1_desk	0.1030	0.0918	0.0647
TUM fr2_desk	0.0641	0.0724	0.0598
TUM fr3_office	0.0686	0.0531	0.0996

C. Advanced Sensors

Mobile robotics and various applications of SLAM, convolutional neural networks (CNN), and VR/AR are constrained by power resources and low frame rates. These applications can not only benefit from high frame rate, but also could save resources if they consumed less energy.

Monocular cameras have been used in many scene understanding and SLAM algorithms [37]. Passive stereo cameras (e.g., Bumblebee2, 48 FPS @ 2.5 W [48]), structured light cameras (e.g., Kinect, 30 FPS @ 2.25 W [49]) and Time-of-flight cameras (e.g., Kinect One, 30 FPS @ 15 W [49]) additionally provide metric depth measurements; however, these cameras are limited by low frame rate and have relatively demanding power budget for mobile devices, problems that modern bioinspired and analogue methods are trying to address.

Dynamic vision sensor (DVS), also known as the event camera, is a novel bioinspired imaging technology, which has the potential to address some of the key limitations of conventional imaging systems. Instead of capturing and sending a full frame, an event camera captures and sends a set of sparse *events*, generated by the change in the intensity. They are low power and are able to detect changes very quickly. Event cameras have been used in camera tracking [50], optical flow estimation [51], and pose estimation [52]– [54]. Very high dynamic range of DVS makes it suitable for real-world applications.

Cellular vision chips, such as the ACE400 [55], ACE16K [56], MIPA4K [57], and focal-plane sensor-processor arrays (FPSPs) [58]–[60], integrate sensing and processing

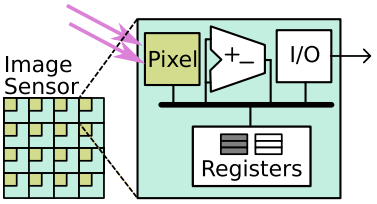

Fig. 6. *Focal-plane sensor-processor arrays (FPSPs) are parallel processing systems, where each pixel has a processing element.*

in the focal plane. FPSPs are massively parallel processing systems on a single chip. By eliminating the need for data transmission, not only is the effective frame rate increased, but also the power consumption is reduced significantly. The individual processing elements are small general purpose analogue processors with a reduced instruction set and memory. Fig. 6 shows a concept diagram of FPSP, where each pixel not only has a light-sensitive sensor, but also has a simple processing element. The main advantages of FPSPs are the high effective frame rates at lower clock frequencies, which in turn reduces power consumption compared to conventional sensing and processing systems [61]. However, with the limited instruction sets and local memory [60], developing new applications for FPSPs, such as image filtering or camera tracking, is a challenging problem.

In the past, several interesting works have been presented using FPSPs, including high-dynamic range imaging [62]. New directions are being followed to explore the performance of FPSPs in real-world robotic and virtual reality applications. These directions include: 1) 4-DOF camera tracking [63] and 2) automatic filter kernel code generation as well as Viola-Jones [64] face detection [65]. The key concept behind these works with FPSP is the fact that FPSP is able to report sum of intensity values of all (or a selection of) pixels in just one clock cycle. This ability allows us to develop kernel code generation and also develop/verify motion hypotheses for visual odometry and camera tracking applications. The results of these works demonstrate that FPSPs not only consume much less power compared to conventional cameras, but also can be operated at very high frame rates, such as 10 000 FPS.

Fig. 7 demonstrates execution times for common convolution filters on various CPUs and GPUs compared with an implementation of FPSP, known as SCAMP [60]. The code for FPSP was automatically generated as explained in [65]. The parallel nature of the FPSP allows it to perform all of the tested filter kernels, shown on x-axis, in a fraction of the time needed by the other devices, shown on y-axis. This is a direct consequence of having a dedicated processing element available for every pixel, building up the filter on the whole image at the same time. As for the other devices, we see that for dense kernels (*Gauss, Box*), GPUs usually perform better than CPUs, whereas for sparse kernels (*Sobel, Laplacian, Sharpen*), CPUs seem to have an advantage. An outlier case is the 7×7 box filter, at which only the most powerful graphics card manages to get a result comparable to the CPUs. It is assumed that the CPU implementation follows a more suitable algorithm than the GPU implementation, even though both implementations are based on their vendors performance libraries (*Intel IPP, nVidia NPP*). Another reason could be the fact that the *GTX680* and *GTX780* are based on a hardware architecture that is less suitable for this type of filter than the *TITAN X*'s architecture. While Fig. 7 shows that there is a significant reduction in *execution time*, the SCAMP chip consumes only $1.23W$ under full load. Compared to the experimented CPU and GPU systems, this is at least 20 times less *power*. Clearly, a more specialized image processing pipeline architecture could be more energy-efficient than these fully programmable architectures. There is scope for further research to map the space of alternative designs, including specialized heterogeneous multicore vision processing accelerators such as the Myriad-2 vision processing unit [66].

III. SOFTWARE OPTIMIZATIONS, COMPILERS AND RUNTIMES

In this section, we investigate how software optimizations, that are mainly implemented as a collection of compiler and runtime techniques, can be used to deliver potential improvements in power consumption and speed tradeoffs. The optimizations must determine how to efficiently map and schedule program parallelism onto multicore, heterogeneous processor architectures. This section presents the novel *static*, *dynamic*, and *hybrid* approaches used to specialize computer vision applications for execution on energy efficient runtimes and hardware (Fig. 8).

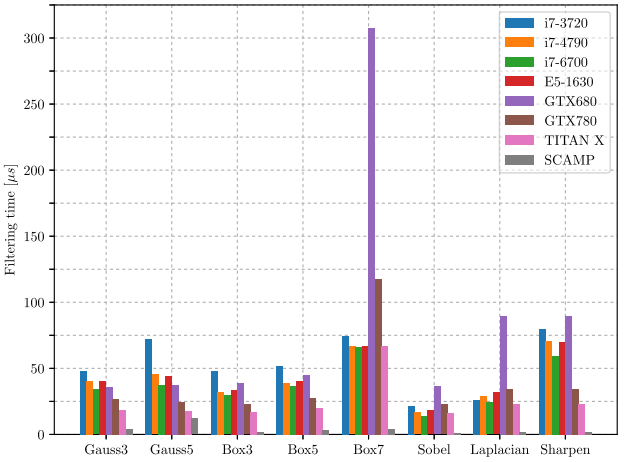

Fig. 7. *Time for a single filter application of several well-known filters on CPU, GPU, and SCAMP FPSP hardware. The FPSP code was generated by the method explained in [65], the CPU and GPU code are based on OpenCV 3.3.0.*

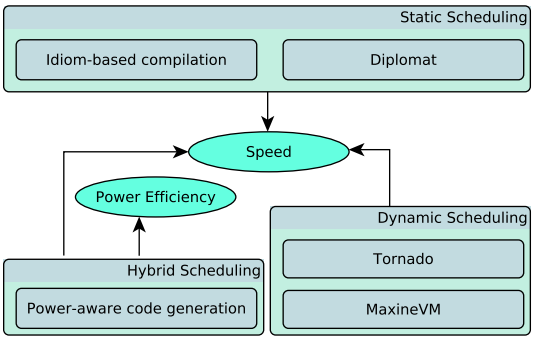

Fig. 8. ***Static, dynamic, and hybrid scheduling are the software optimization methods presented for power efficiency and speed improvement.***

A. Static Scheduling and Code Transformation

In this section, we focus on static techniques applied when building an optimized executable. Static schedulers and optimizers can only rely on performance models of underlying architectures or code to optimize, which limit opportunities. However, they do not require additional code to execute, which reduces runtime overhead. We first introduce in Section III-A1 an idiom-based heterogeneous compilation methodology which given the source code of a program, can automatically identify and transform portions of code in order to be accelerated using many-core CPUs or GPUs. Then, in Section III-A2, we propose a different methodology used to determine which resources should be used to execute those portions of code. This methodology takes a specialized direction, where applications need to be expressed using a particular model in order to be scheduled.

1) Idiom-Based Heterogeneous Compilation: A wide variety of high-performance accelerators now exist, ranging from embedded DSPs, to GPUs, to highly specialized devices such as tensor processing unit [67] and vision processing unit [66]. These devices have the capacity to deliver high performance and energy efficiency, but these improvements come at a cost: to obtain peak performance, the target application or kernel often needs to be rewritten or heavily modified. Although high-level abstractions can reduce the cost and difficulty of these modifications, these make it more difficult to obtain peak performance. In order to extract the maximum performance from a particular accelerator, an application must be aware of its exact hardware parameters [number of processors, memory sizes, bus speed, network-on-chip (NoC) routers, etc.], and this often requires low level programming and tuning. Optimized numeric libraries and domain specific languages (DSLs) have been proposed as a means of reconciling programmer ease and hardware performance. However, they still require significant legacy code modification and increase the number of languages programmers need to master.

Ideally, the compiler should be able to automatically take advantage of these accelerators, by identifying opportunities for their use, and then automatically calling into the appropriate libraries or DSLs. However, in practice, compilers struggle to identify such opportunities due to the complex and expensive analysis required. Additionally, when such opportunities are found, they are frequently on a too small scale to obtain any real benefit, with the cost of setting up the accelerator [i.e., data movement, remote procedure call (RPC) costs, etc.] being much greater than the improvement in execution time or power efficiency. Larger scale opportunities are difficult to identify due to the complexity of analysis, which often requires interprocedural analyses, loop invariant detection, pointer and alias analyses, etc., which are complex to implement in the compiler and expensive to compute. On the other hand, when humans attempt to use these accelerators, they often lack the detailed knowledge of the compiler and resort to "hunches" or *ad hoc* methods, leading to suboptimal performance.

In [68], we develop a novel approach to automatically detect and exploit opportunities to take advantage of accelerators and DSLs. We call these opportunities "idioms." By expressing these idioms as constraint problems, we can take advantage of constraint solving techniques [in our case a satisfiability modulo theories (SMTs) solver]. Our technique converts the constraint problem which describes each idiom into an LLVM compiler pass. When running on LLVM intermediate representation (IR), these passes identify and report instances of each idiom. This technique is further strengthened by the use of symbolic execution and static analysis techniques, so that formally proved transformations can be automatically applied when idioms are detected.

We have described idioms for sparse and dense linear algebra, and stencils and reductions, and written transformations from these idioms to the established cuSPARSE and clSPARSE libraries, as well as a data-parallel, functional DSL which can be used to generate high performance platform specific OpenCL code. We have then evaluated this technique on the NAS, Parboil, and Rodinia sequential C/C++ benchmarks, where we detect 55 instances of our described idioms. The NAS, Parboil, and Rodinia benchmarks include several key and frequently used computer vision and SLAM related tasks such as convolution filtering, particle filtering, backpropagation, k-means clustering, breadth-first search, and other fundamental computational building blocks. In the cases where these idioms form a significant part of the sequential execution time, we are able to transform the program to obtain performance improvements ranging from 1.24x to over 20x on integrated and discrete GPUs, contributing to the fast *execution time* objective.

2) Diplomat, Static Mapping of Multikernel Applications on Heterogeneous Platforms: We propose a novel approach to heterogeneous embedded systems programmability using

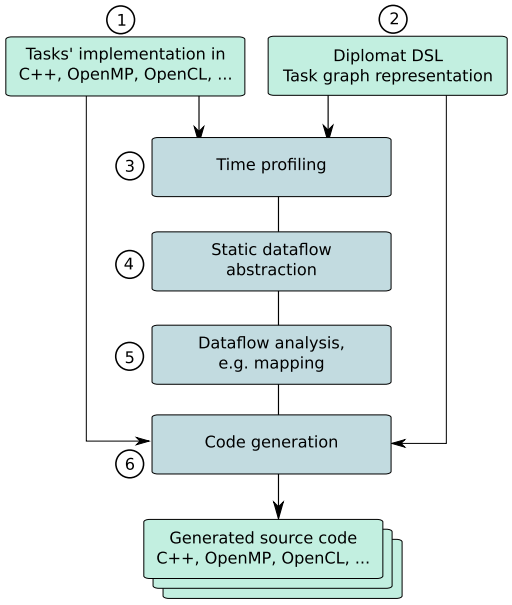

Fig. 9. ***An overview of the Diplomat framework. The user provides (1) the task implementations in various languages and (2) the dependencies between the tasks. Then in (3) Diplomat performs timing analysis on the target platform and in (4) abstracts the task-graph as a static dataflow model. Finally, a dataflow model analysis step is performed in (5), and in (6) the Diplomat compiler performs the code generation.***

a task-graph based DSL called Diplomat [69]. Diplomat is a task-graph framework that exploits the potential of static dataflow modelling and analysis to deliver performance estimation and CPU/GPU mapping. An application has to be specified once, and then the framework can automatically propose good mappings. This work aims at improving runtime as much as performance robustness.

The Diplomat front-end is embedded in the Python programming language and it allows the framework to gather fundamental information about the application: the different possible implementations of the tasks, their expected input and output data sizes, and the existing data dependencies between each of them.

At compile-time, the framework performs static analysis. In order to benefit from existing dataflow analysis techniques, the initial task-graph needs to be turned into a dataflow model. As the dataflow graph will not be used to generate the code, a representation of the application does not need to be precise. But it needs to model an application's behavior close enough to obtain good performance estimations. Diplomat performs the following steps. First, the initial task-graph is abstracted into a static dataflow formalism. This includes a timing profiling step to estimate task durations and communication delays. Then, by using static analysis techniques [70], a throughput evaluation and a mapping of the application are performed.

Once a potential mapping has been selected, an executable C++ code is automatically generated. This generated implementation takes advantage of task-parallelism and data-parallelism. It can use OpenMP and OpenCL and it may apply partitioning between CPU and GPU when it is beneficial. This overview is summarized in Fig. 9.

We evaluate Diplomat with KinectFusion on two embedded platforms, Odroid-XU3 and Arndale, with four different configurations for algorithmic parameters, chosen manually. Fig. 10 shows the results for Arndale for four different configurations, marked as ARN0...3. Using Diplomat, we observed a 16% speed improvement on average and up to a 30% improvement over the best existing hand-coded implementation. This is an improvement on *runtime speed*, one of the goals outlined earlier.

B. Dynamic Scheduling

Dynamic scheduling takes place while the optimized program runs with actual data. Because dynamic schedulers can monitor actual performance, they can compensate for performance skews due to data-dependant control-flow and computation that static schedulers cannot accurately capture and model. Dynamic schedulers can therefore exploit additional dynamic runtime information to enable more optimization opportunities. However, they also require the execution of additional profiling and monitoring code, which can create performance penalties.

Tornado and MaxineVM runtime scheduling are research prototype systems that we are using to explore and investigate dynamic scheduling opportunities. Tornado is a framework (prototyped on top of Java) using dynamic scheduling for transparent exploitation of task-level parallelism on heterogeneous systems having multicore CPUs, and accelerators such as GPUs, DSPs and FPGAs. MaxineVM is a research Java Virtual Machine (JVM) that we are initially using to investigate dynamic heterogeneous multicore scheduling for application and JVM service threads in order to better meet the changing

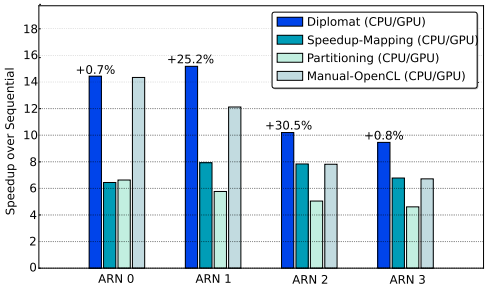

Fig. 10. ***Evaluation of the best result obtained with Diplomat for CPU and GPU configurations, and comparison with handwritten solutions (OpenMP, OpenCL) and automatic heuristics (Partitioning, Speed-up mapping) for KinectFusion on Arndale platform. The associated numbers on x-axis are different KinectFusion algorithmic parameter configuration, and the percent on top of Diplomat bars are the speedup over the manual implementation.***

power and performance objectives of a system under dynamically varying battery life and application service demands.

1) Tornado: Tornado is a heterogeneous programming framework that has been designed for programming systems that have a higher degree of heterogeneity than existing GPGPU accelerated systems and where system configurations are unknown until runtime. The current Tornado prototype [71] superseding JACC, described in [72], can dynamically offload code to big.LITTLE cores, and GPUs with its OpenCL backend that supports the widest possible set of accelerators. Tornado can also be used to generate OpenCL code that is suitable for high-level synthesis tools in order to produce FPGA accelerators, although it is not practical to do this unless the relatively long place and route times of FPGA vendor tools can be amortized by application runtime overheads. The main benefit of Tornado is that it allows portable dynamic exploration of how heterogeneous scheduling decisions for task-parallel frameworks will lead to improvements in power-performance tradeoffs without rewriting the application level code, and also where knowledge of the heterogeneous configuration of a system is delayed until runtime.

The Tornado API cleanly separates computation logic from coordination logic that is expressed using a task-based programming model. Currently, data parallelization is expressed using standard Java support for annotations [71]. Applications remain architecture-neutral, and as the current implementation of Tornado is based on the Java managed language, we are able to dynamically generate code for heterogeneous execution without recompilation of the Java source, and without manually generating new optimized routines for any accelerators that may become available. Applications need only to be configured at runtime for execution on the available hardware. Tornado currently uses an OpenCL driver for maximum device coverage: this includes mature support for multicore CPUs and GPGPU, and maturing support for Xeon Phi coprocessor/accelerators. The current dynamic compiler technology of Tornado is built upon JVMCI and GRAAL APIs for Java 8 and above. The sequential Java and C++ versions of KinectFusion in SLAMBench both perform at under 3 FPS with the C++ version being 3.4x faster than Java. This improvement of *runtime speed* is shown in Fig. 11. By accelerating KinectFusion through GPGPU execution using Tornado, we manage to achieve a constant rate of over 30 FPS (33.13 FPS) across all frames (882) from the ICL-NUIM data set with room 2 configuration [28]. To achieve 30 FPS, all kernels have been accelerated by up to 821.20x with an average of 47.84x across the whole application [71], [73]. Tornado is an attractive framework for the development of portable computer vision applications as its dynamic JIT compilation for traditional CPU cores and OpenCL compute devices such as GPUs enables real-time performance constraints to be met while eliminating the need to rewrite and optimize code for different GPU devices.

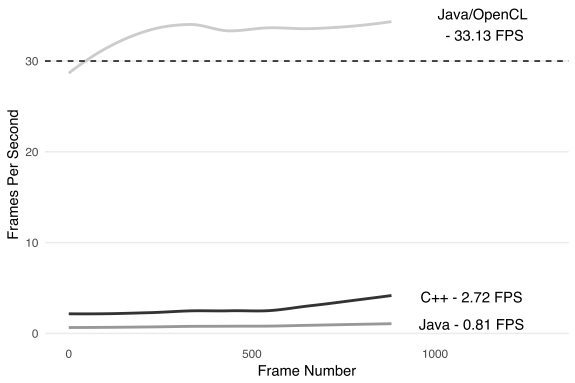

Fig. 11. ***Execution performance of KinectFusion (using FPS) over the time using Tornado (Java/OpenCL) vs. baseline Java and C++.***

2) MaxineVM: The main contribution of MaxineVM is to provide a research infrastructure for managed runtime systems that can execute on top of modern instruction set architectures (ISAs) supplied by both Intel and ARM. This is especially relevant because ARM is the dominant ISA in mobile and embedded platforms. MaxineVM has been released as open-source software [74].

Heterogeneous multicore systems comprised of CPUs having the same ISA but different power/performance design point characteristics create a significant challenge for virtual machines that are typically agnostic to CPU core heterogeneity when undertaking thread-scheduling decisions. Further, heterogeneous CPU core clusters are typically attached to NUMA-like memory system designs, consequently thread scheduling policies need to be adjusted to make appropriate decisions that do not adversely affect the performance and power consumption of managed applications.

In MaxineVM, we are using the Java managed runtime environment to optimize thread scheduling for heterogeneous architectures. Consequently, we have chosen to use and extend the Oracle Labs research project software for MaxineVM [75] that provided a state-of-the-art research VM for x86–64. We have developed a robust port of MaxineVM to ARMv7 [71], [76] (an AArch64 port is also in progress) ISA processors that can run important Java and SLAM benchmarks, including a Java version of KinectFusion. MaxineVM has been designed for maximum flexibility, this sacrifices some performance, but it is trivially possible to replace the public implementation of an interface or *scheme*, such as for monitor or garbage collection with simple command line switches to the command that generates a MaxineVM executable image.

C. Hybrid Scheduling

Hybrid scheduling considers dynamic techniques, taking advantage of static and dynamic data. A schedule can be

statically optimized for a target architecture and application (i.e., using machine learning), and a dynamic scheduler can further adjust this schedule to optimize further actual code executions. Since it can rely on a statically optimized schedule, the dynamic scheduler can save a significant amount of work and therefore lower its negative impact on performance.

1) Power-Aware Code Generation: Power is an important constraint in modern multicore processor design. We have shown that power across heterogeneous cores varies considerably [77]. This work develops a compiler-based approach to runtime power management for heterogeneous cores. Given an externally determined power budget, it generates parallel parameterized partitioned code that attempts to give the best performance within that power budget. It uses the compiler infrastructure developed in [78]. The hybrid scheduling has been tested on standard benchmarks such as DSPstone, UTSDP, and Polybench. These benchmarks provide an in-depth comparison with other methods and include key building blocks of many SLAM and computer vision tasks such as matrix multiplication, edge detection, and image histogram. We applied this technique to embedded parallel OpenMP benchmarks on the TI OMAP4 platform for a range of power budgets. On average we obtain a 1.37x speed-up over dynamic voltage and frequency scaling (DVFS). For low power budgets, we see a 2x speed-up improvement. SLAM systems, and vision applications in general, are composed of different phases. An adaptive power budget for every phase positively impacts frame rate and power consumption.

IV. HARDWARE AND SIMULATION

The designers of heterogeneous multiprocessor system-on-chip (MPSoC) are faced with an enormous task when attempting to design a system that is cooptimized to deliver power-performance efficiency under a wide range of dynamic operating conditions concerning the available power stored in a battery and the current application performance demands. In this paper, a variety of simulation tools and technologies have been presented to assist designers in their evaluations of how performance, energy, and power consumption tradeoffs are affected by computer vision algorithm parameters and computational characteristics of specific implementations on different heterogeneous processors and accelerators. Tools have been developed that focus on the evaluation of native and managed runtime systems, that execute on ARM and x86–64 processor instruction set architectures in conjunction with GPU and custom accelerator intellectual property.

The contributions of this section have been organized under three main topics: *simulation*, *profiling*, and *specialization*. Under each topic, several novel tools and methods are presented. The main objective in developing these tools and methods is to reduce development complexity and increase reproducibility for system analysis. Fig. 12 presents a graph where all simulation, profiling, and specialization tools are summarized.

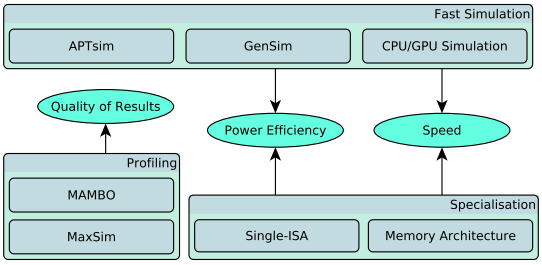

Fig. 12. ***Hardware development tasks are simulation, profiling, and specialization tools, each with its own goals. With these three tasks, it is possible to develop customized hardware for computer vision applications.***

A. Fast Simulation

Simulators have become an essential tool for hardware design. They allow designers to prototype different systems before committing to a silicon design, and save enormous amounts of money and time. They allow embedded systems engineers to develop the driver and compiler stack, before the system is available, and to be able to verify their results. Even after releasing the hardware, software engineers can make use of simulators to prototype their programs in a virtual environment, without the latency of flashing the software onto the hardware, or even without access to the hardware.

These different use cases require very different simulation technologies. Prototyping hardware typically requires "detailed" performance modelling simulation to be performed, which comes with a significant slowdown compared to real hardware. On the other hand, software development often does not require such detailed simulation, and so faster "functional" simulators can be used. This has led to the development of multiple simulation systems within this work, with the GenSim system being used for "functional" simulation and APTsim being used for more detailed simulation.

In this section, three novel system simulation works are presented. These works are: GenSim, CPU/GPU simulation, and APTsim.

1) The GenSim Architecture Description Language: Modern CPU architectures often have a large number of extensions and versions. At the same time, simulation technologies have improved, making simulators both faster and more accurate. However, this has made the creation of a simulator for a modern architecture much more complex. Architecture description languages (ADLs) seek to solve this problem by decoupling the details of the simulated architecture from the tool used to simulate it.

We have developed the GenSim simulation infrastructure, which includes an ADL toolchain (see Fig. 13).

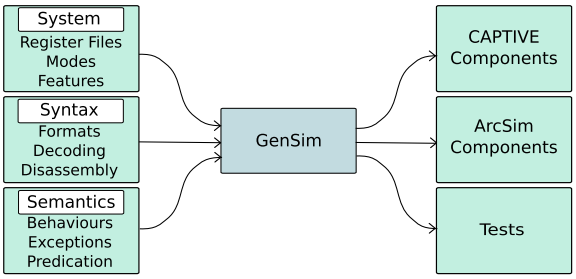

Fig. 13. ***Diagram showing the general flow of the GenSim ADL toolchain.***

This ADL is designed to enable the rapid development of fast functional simulation tools [79], and the prototyping of architectural extensions (and potentially full instruction set architectures). This infrastructure is used in the CPU/GPU simulation work (Section IV-A2). The GenSim infrastructure is described in a number of publications [80]–[82]. GenSim is available under a permissive open-source license and is available at [83].

2) Full-System Simulation for CPU/GPU: Graphics processing units are highly specialized processors that were originally designed to process large graphics workloads effectively; however, they have been influential in many industries, including in executing computer vision tasks. Simulators for parallel architectures, including GPUs, have not reached the same level of maturity as simulators for CPUs, both due to the secrecy of leading GPU vendors, and the problems arising from mapping parallel onto scalar architectures, or onto different parallel architectures.

At the moment, GPU simulators that have been presented in literature have limitations, resulting from lack of verification, poor accuracy, poor speeds, and limited observability due to incomplete modelling of certain hardware features. As they do not accurately model the full native software stack, they are unable to execute realistic GPU workloads, which rely on extensive interaction with user and system runtime libraries.

In this work, we propose a full-system methodology for GPU simulation, where rather than simulating the GPU as an independent unit, we simulate it as a component of a larger system, comprising a CPU simulator with supporting devices, operating system, and a native, unmodified driver stack. This faithful modelling results in a simulation platform indistinguishable from real hardware.

We have been focusing our efforts on simulation of the ARM Mali GPU and have built a substantial amount of surrounding infrastructure. We have seen promising results in simulation of compute applications, most notably SLAMBench.

The work directly contributed to full system simulation, by implementing the ARMv7 MMU, ARMv7 and Thumb-2 instruction sets, and a number of devices needed to communicate with the GPU. To connect the GPU model realistically, we have implemented an ARM CPU GPU interface containing an ARM Device on the CPU side [84].

The implementation of the Mali GPU simulator comprises:

- implementation of the job manager, a hardware resource for controlling jobs on the GPU side;
- shader core infrastructure, which allows for retrieving important context, needed to execute shader programs efficiently;
- shader program decoder, which allows us to interpret Mali Shader binary programs;
- shader program execution engine, which allows us to simulate the behavior of Mali programs.

Future plans for simulation include extending the infrastructure to support real time graphics simulation, increasing GPU Simulation performance using dynamic binary translation (DBT) [79], [82], [85] techniques, and extending the Mali model to support performance modelling. We have also continued to investigate new techniques for full-system dynamic binary translation (such as exploiting hardware features on the host to further accelerate simulation performance), as well as new methodologies for accelerating the implementation and verification of full system instruction set simulators. Fast full system simulation presents a large number of unique challenges and difficulties and in addressing and overcoming these difficulties, we expect to be able to produce a significant body of novel research. Taken as a whole, these tools will directly allow us to explore next-generation many-core applications, and design hardware that is characterized by high performance and low power.

3) APTSim–Simulation and Prototyping Platform: APTSim (Fig. 14) is intended as a fast simulator allowing rapid simulation of microprocessor architectures and microarchitectures as well as the prototyping of accelerators. The system runs on a platform consisting of a processor, for functional simulation, and an FPGA for implementing architecture timing models and prototypes. Currently the Xilinx Zynq family is used as the host platform. APTSim performs dynamic binary instrumentation using MAMBO; see Section IV-B1, to dynamically instrument a running executable along with the MAST codesign library, described as follows. Custom MAMBO plugins allow specific instructions, such as load/store or PC changing events to be sent to MAST hardware models, such as memory systems or processor pipeline. From a simulation perspective the hardware models are for timing and gathering statistics and do not perform functional simulation, which is carried out on the host processor as native execution; so, for example, if we send a request to a cache system the model will tell us at which memory level the result is present in as well as a response time, while the actual data will be returned from the processor's own memory. This separation allows for smaller, less complicated, hardware models to gather statistics while the processor executes the

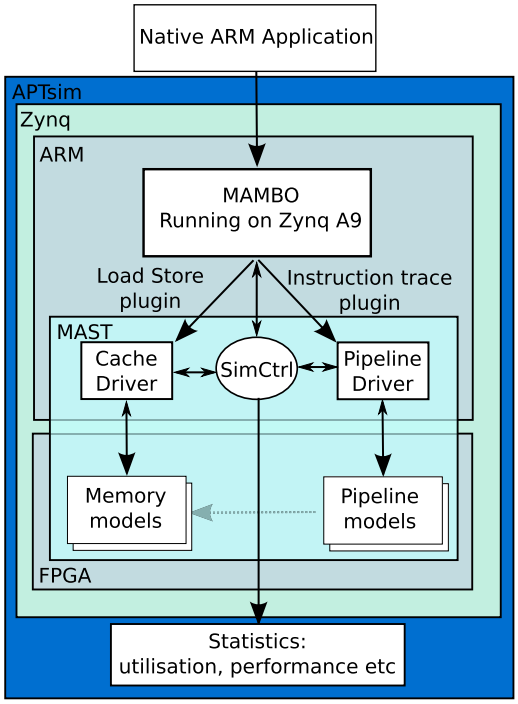

Fig. 14. ***APTSim an FPGA accelerated simulation and prototyping platform, currently implemented on Zynq SoC.***

benchmark natively and the MAMBO plugins capture the necessary events with low overhead.

The MAST library provides a framework for easily integrating many hardware IP blocks, implemented on FPGA, with a linux-based application running on a host processor. MAST consists of two principal parts: a software component and a hardware library. The software component allows the discovery and management of hardware IP blocks and the management of memory used by the hardware blocks; critically this allows new hardware blocks to be configured and used at runtime using only user space software. The hardware library, written in Bluespec, contains parameterized IP blocks including architecture models such as cache systems or pipeline models and accelerator modules for computer vision, such as filters or feature detectors. The hardware models can either be masters or slaves, from a memory perspective. As masters, models can directly access processor memory leaving the processor to execute code while the hardware is analyzing the execution of the last code block.

APTSim also allows us to evaluate prototype hardware, for example we evaluated multiple branch predictors by implementing them in Bluespec and using a MAST compliant interface. This allows us to execute our benchmark code once on the CPU and offload to multiple candidate implementations to rapidly explore the design space.

In [86], we show that on the Xilinx Zynq 7000 FPGA board coupled with a relatively slow 666 MHz ARM9 processor, the slowdown of APTsim is 400x in comparison to native execution on the same processor. While a relatively important slowdown over native execution is unavoidable to implement a fine performance monitoring, slowdown on APTsim is about half of GEM5 running at 3.2 GHz on an Intel Xeon E3 to simulate the same ARM system. Note that, contrary to APTsim, GEM5 on Xeon does not take profit of any FPGA acceleration. This shows the interest of APTsim to take profit of FPGA acceleration to implement a fast register transfer level (RTL) simulation and monitor its performance, while hiding the complexity of FPGA programming from the user.

B. Profiling

Profiling is the process of analyzing the runtime behavior of a program in order to perform some measurements about the performance of the program, for example, to determine which parts of the program take the most time to execute. This information can then be used to improve software (for example, by using a more optimized implementation of frequently executed functions) or to improve hardware (by including hardware structures or instructions which provide better performance for frequently executed functions). Profiling of native applications is typically performed via dynamic binary instrumentation. However, when a managed runtime environment is used, the runtime environment can often perform the necessary instrumentation. In this section, we explore both of these possibilities, with MAMBO being used for native profiling, and MaxSim being used for the profiling of Java applications.

1) MAMBO: Instruction Level Profiling: Dynamic Binary Instrumentation (DBI) is a technique for instrumenting applications transparently while they are executed, working at the level of machine code. As the ARM architecture expands beyond its traditional embedded domain, there is a growing interest in DBI systems for the general-purpose multicore processors that are part of the ARM family. DBI systems introduce a performance overhead and reducing it is an active area of research; however, most efforts have focused on the x86 architecture.

MAMBO is a low overhead DBI framework for 32-bit (AArch32) and 64-bit ARM (AArch64) [87]. MAMBO is open-source [88]. MAMBO provides an event-driven plugin API for the implementation of instrumentation tools with minimal complexity. The API allows the enumeration, analysis and instrumentation of the application code ahead of execution, as well as tracking and control of events such as system calls. Furthermore, the MAMBO API provides a number of high level facilities for developing portable instrumentation, i.e., plugins which can execute efficiently both on AArch32 and AArch64, while being implemented using mostly high level architecture-agnostic code.

MAMBO incorporates a number of novel optimizations, specifically designed for the ARM architecture, which allow

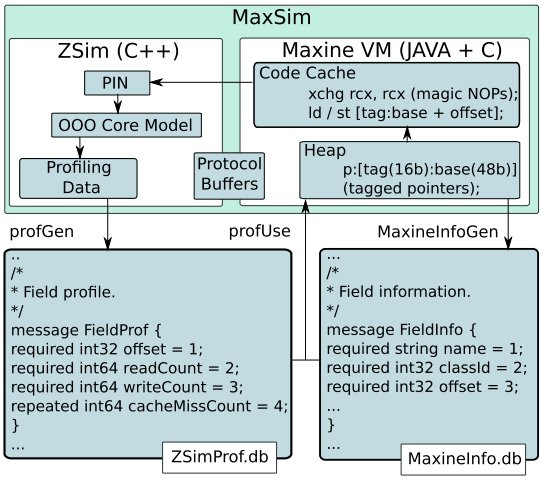

Fig. 15. ***MaxSim overview of Zsim and MaxineVM based profiling.***

it to minimize its performance overhead. The geometric mean runtime overhead of MAMBO running SPEC CPU2006 with no instrumentation is as low as 12% (on an APM X-C1 system), compared DynamoRIO [89], a state of the art DBI system, which has an overhead of 34% under the same test conditions.

2) MaxSim: Profiling and Prototyping Hardware–Software Codesign for Managed Runtime Systems: Managed applications, written in programming languages such as Java, C# and others, represent a significant share of workloads in the mobile, desktop, and server domains. Microarchitectural timing simulation of such workloads is useful for characterization and performance analysis, of both hardware and software, as well as for research and development of novel hardware extensions. MaxSim [90] (see Fig. 15) is a simulation platform based on the MaxineVM [75] (explained in Section III-B2), the ZSim [91] simulator, and the McPAT [92] modelling framework. MaxSim can perform fast and accurate simulation of managed runtime workloads running on top of Maxine VM [74]. MaxSim's capabilities include: 1) low-intrusive microarchitectural profiling via pointer tagging on x86–64 platforms; 2) modeling of hardware extensions related, but not limited to, tagged pointers; and 3) modelling of complex software changes via address-space morphing.

Low-intrusive microarchitectural profiling is achieved by utilizing tagged pointers to collect type- and allocation-site related hardware events. Furthermore, MaxSim allows, through a novel technique called address space morphing, the easy modelling of complex object layout transformations. Finally, through the codesigned capabilities of MaxSim, novel hardware extensions can be implemented and evaluated. We showcase MaxSim's capabilities by simulating the whole set of the DaCapo-9.12-bach benchmarks in less than a day while performing an up-to-date microarchitectural power and performance characterization [90]. Furthermore, we demonstrate a hardware/software codesigned optimization that performs dynamic load elimination for array length retrieval achieving up to 14% L1 data cache loads reduction and up to 4% dynamic energy reduction. In [93] we present results for MaxineVM with MaxSim. We use SLAMBench to experiment with KinectFusion on a 4-core Nehalem system, using 1 and 4 cores (denoted by *1C* and *4C*, respectively). We use MaxSim's extensions for the address generation unit (AGU) (denoted by 1CA and 4CA) and load-store unit (LSU) extension (shown by 1CAL and 4CAL). Fig. 16(a) shows heap savings of more than 30% on SLAMBench thanks to class information pointer (CIP) elimination. Fig. 16 (b) demonstrates the relative reduction in execution time, using the proposed framework. In this figure, *EA* refers to a machine configuration with CIP elimination with 16-b CID (class information) and *EAL* refers to a variant with CIP elimination, 4 bits CID, and AGU and LSU extensions. B stands for the standard baseline MaxSim virtual machine and C is B with object compression. Fig. 16(b) shows up to 6% execution time performance benefits of CIP elimination over MaxSim with none of our extension, whether its uses 4 cores (*4CA-EA/4CA-B*) or 1 core (*1CA-EA/1C-B*) Finally, Fig. 16 (c) shows the relative reduction in DRAM dynamic energy for the cases mentioned above. As the graph shows, there is an 18% to 28% reduction in DRAM dynamic energy. These reductions contribute to the objective of having improved quality of the results. MaxSim is open-source [74].

C. Specialization

Recent developments in computer vision and machine learning have challenged hardware and circuit designers to design faster and more efficient systems for these tasks [94]. TPU from Google [67], VPU from Intel Movidius [66], and IPU from Graphcore [95], are such devices with major reengineerings in hardware design, resulting in outstanding performance. While the development of

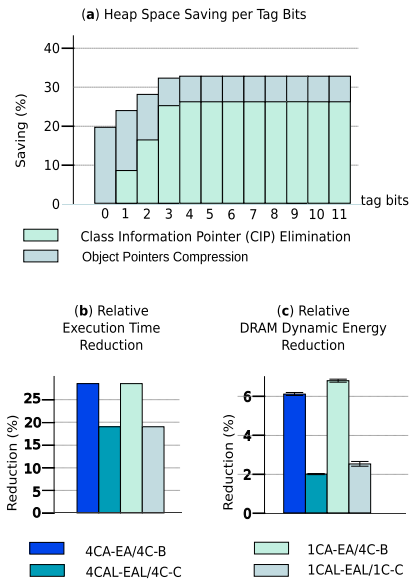

Fig. 16. ***Performance of MaxSim on KinectFusion: (a) heap space saving using tagged pointers; (b) relative reduction in execution time; and (c) relative reduction in DRAM dynamic energy.***

custom hardware can be appealing due to the possible significant benefits, it can lead to extremely high design, development, and verification costs, and a very long time to market. One method of avoiding these costs while still obtaining many of the benefits of custom hardware is to specialize existing hardware. We have explored several possible paths to specialization, including specialized memory architectures for GPGPU computations (which are frequently used in computer vision algorithm implementations), the use of single-ISA heterogeneity (as seen in ARM's big.LITTLE platforms), and the potential for power and area savings by replacing hardware structures with software.

1) Memory Architectures for GPGPU Computation: Current GPUs are no longer perceived as accelerators solely for graphic workloads and now cater to a much broader spectrum of applications. In a short time, GPUs have proven to be of substantive significance in the world of general-purpose computing, playing a pivotal role in scientific and high performance computing (HPC). The rise of general-purpose computing on GPUs has contributed to the introduction of on-chip cache hierarchies in those systems. Additionally, in SLAM algorithms, reusing previously processed data frequently occurs such as in bundle adjustment, loop detection, and loop closure. It has been shown that efficient memory use can improve the runtime speed of the algorithm. For instance, the Distributed Particle (DP) filter optimizes memory requirements using an efficient data structure for maintaining the map [96].

We have carried out a workload characterization of GPU architectures on general-purpose workloads, to assess the efficiency of their memory hierarchies [97] and proposed a novel cache optimization to resolve some of the memory performance bottlenecks in GPGPU systems [98].

In our workload characterization study (overview on Fig. 17) we saw that, in general, high level-1 (L1) data cache miss rates place high demands on the available level-2 (L2) bandwidth that is shared by the large number of cores in typical GPUs. In particular, Fig. 17 represents bandwidth as the number of instructions per cycle (IPCs). Furthermore, the high demand for L2 bandwidth leads to extensive congestion in the L2 access path, and in turn this leads to high memory latencies. Although GPUs are heavily multithreaded, in memory intensive applications the memory latency becomes exposed due to a shortage of active compute threads, reducing the ability of the multithreaded GPU to hide memory latency (exposed latency range on Fig. 17). Our study also quantified congestion in the memory system, at each level of the memory hierarchy, and characterized the implications of high latencies due to congestion. We identified architectural parameters that play a pivotal role in memory system congestion, and explored the design space of architectural options to mitigate the bandwidth bottleneck. We showed that the improvement in performance achieved by mitigating the bandwidth bottleneck in the cache hierarchy can exceed the speedup obtained by simply increasing the on-chip DRAM bandwidth. We also showed that addressing the bandwidth bottleneck in isolation at specific levels can be suboptimal and can even be counterproductive. In summary, we showed that it is imperative to resolve the bandwidth bottleneck synergistically across all levels of the memory hierarchy. The second part of our work in this area aimed to reduce the pressure on the shared L2 bandwidth. One of the key factors we have observed is that there is significant replication of data among private L1 caches, presenting an opportunity to reuse data among the L1s. We have proposed a cooperative caching network (CCN), which exploits reuse by connecting the L1 caches with a lightweight ring network to facilitate intercore communication of shared data. When measured on a selection of GPGPU benchmarks, this approach delivers a performance improvement of 14.7% for applications that exhibit reuse.

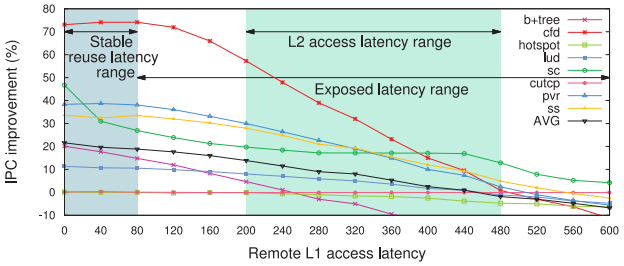

Fig. 17. ***Speed-up of instructions per cycle (IPC) with varying remote L1 access latencies.***

2) Evaluation of Single-ISA Heterogeneity: We have investigated the design of heterogeneous processors sharing a common ISA. The underlying motivation for single-ISA heterogeneity is that a diverse set of cores can enable runtime flexibility. We argue that selecting a diverse set of heterogeneous cores to enable flexible operation at runtime is a nontrivial problem due to diversity in program behavior. We further show that common evaluation methods lead to false conclusions about diversity. We suggest the Kolmogorov–Smirnov (KS) test statistical test as an evaluation metric. The KS test is the first step towards a heterogeneous design methodology that optimizes for runtime flexibility [99], [100].

A major roadblock to the further development of heterogeneous processors is the lack of appropriate evaluation metrics. Existing metrics can be used to evaluate individual cores, but to evaluate a heterogeneous processor, the cores must be considered as a collective. Without appropriate metrics, it is impossible to establish design goals for processors, and it is difficult to accurately compare two different heterogeneous processors. We present four new metrics to evaluate user-oriented aspects of sets of heterogeneous cores: localized nonuniformity, gap overhead, set overhead, and generality. The metrics consider sets rather than individual cores. We use examples to demonstrate each metric and show that the metrics can be used to quantify intuitions about heterogeneous cores [101].

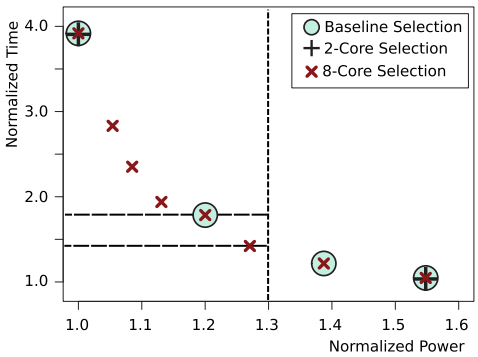

Fig. 18. ***Example of a baseline selection, and 2- and 8-core selections for a specific benchmark application.***

For a heterogeneous processor to be effective, it must contain a diverse set of cores to match a range of runtime requirements and program behaviors. Selecting a diverse set of cores is, however, a nontrivial problem. We present a method of core selection that chooses cores at a range of power-performance points. For example, we see in Fig. 18 that for a normalized power budget of 1.3 (1.3 times higher than the most power-efficient alternative), the best possible normalized time using the baseline selection is 1.75 (1.75 times the fastest execution time), whereas an 8 core selection can lower this ratio to 1.4 without exceeding the normalized power budget, i.e., our method brings a 20% speedup. Our algorithm is based on the observation that it is not necessary for a core to consistently have high performance or low power; one type of core can fulfill different roles for different types of programs. Given a power budget, cores selected with our method provide an average speedup of 7% on EEMBC mobile benchmarks, and a 23% on SPECint 2006 benchmarks over the state of the art core selection method [102].

V. HOLISTIC OPTIMIZATION METHODS

In this section, we introduce holistic optimization methods that combine developments from multiple domains, i.e., hardware, software, and algorithm, to develop efficient end-to-end solutions. The design space exploration work presents the idea of exploring many sets of possible parameters to properly exploit them at different situations. The crowdsourcing further tests the DSE idea on a massive number of devices. Fig. 19 summarizes their goals and contributions.

A. Design Space Exploration

Design space exploration is the exploration of various possible design choices before running the system [103]. In scene understanding algorithms, the possible space of the design choices is very large and spans from high-level algorithmic choices down to parametric choices within an algorithm. For instance, Zhang *et al.* [104] explore algorithmic choices for visual–inertial algorithmic parameters on an ARM CPU, as well as a Xilinx Kintex-7 XC7K355T FPGA. In this section, we introduce two DSE algorithms: The first on, called multidomain DSE, explores algorithmic, compiler and hardware parameters. The second one, coined motion-aware DSE, further adds the complexity of the motion and the environment to the exploration space. The latter work is extended to develop an active SLAM algorithm.

1) Multidomain DSE: Until now, resource-intensive scene understanding algorithms, such as KinectFusion, could only run in real time on powerful desktop GPUs. In [105] we examine how it can be mapped to power constrained embedded systems and we introduce HyperMapper, a tool for multi-objective DSE. HyperMapper was demonstrated in a variety of applications ranging from computer vision and robotics to compilers [44], [105]–[107]. Key to our approach is the idea of incremental codesign exploration, where optimization choices that concern the domain layer are incrementally explored together with low-level compiler and architecture choices (see Fig. 21, dashed boxes). The goal of this exploration is to reduce execution time while minimizing power and meeting our quality of result objective. Fig. 20 shows an example performed with KinectFusion, in which for each point, a set of parameters, two metrics, maximum ATE and runtime speed, is shown. 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 3-D 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.

2) Motion- and Structure-Aware DSE: In multidomain DSE, when tuning software and hardware parameters, we also need to take into account the structure of the environment and the motion of the camera. In the motion- and structure-aware design space exploration (MS-DSE) work [44], we determine the complexity of the structure and motion with a few parameters calculated using information theory. Depending on this complexity and the desired performance metrics, suitable parameters are explored and determined. The hypothesis of MS-DSE is that we

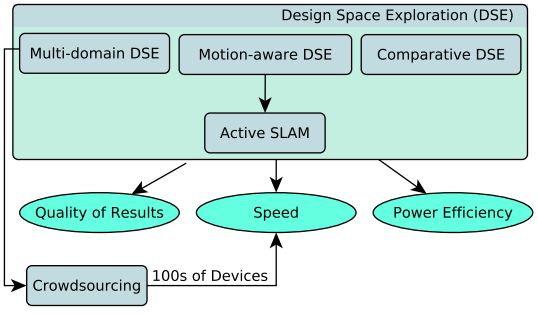

Fig. 19. ***Holistic optimization methods explore all domains of the real-time 3-D scene understanding, including hardware, software, and computer vision algorithms. Two holistic works presented here: Design Space Exploration and Crowdsourcing.***

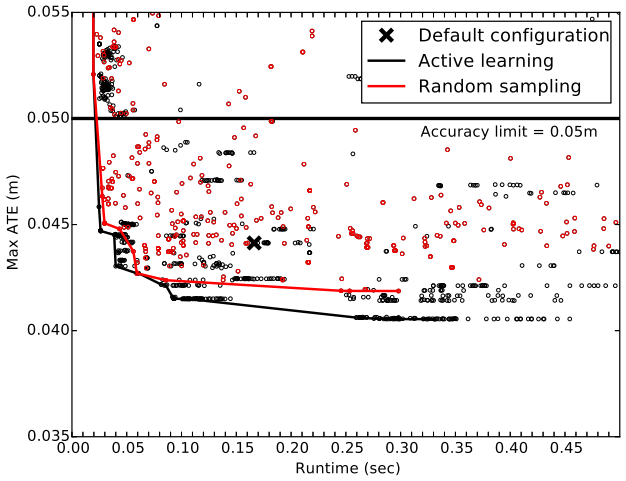

Fig. 20. ***This plot illustrates the result of HyperMapper on the DSE of the KinectFusion algorithmic parameters considering accuracy and frame rate metrics. We can see the result of random sampling (red) as well as the improvement of solutions after active learning (black).***

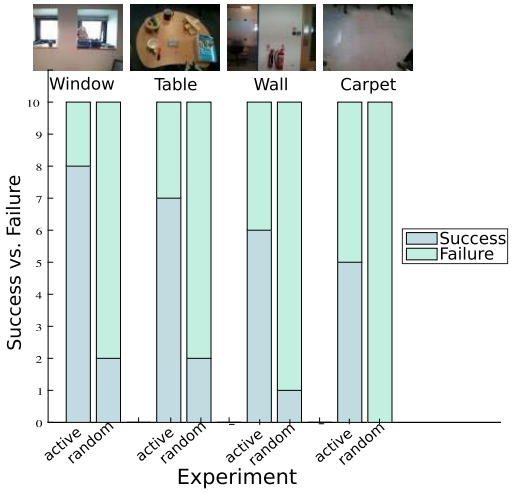

Fig. 22. ***Success versus failure rate when mapping the same environment with different motion planning algorithms: active SLAM and random walk.***

can use a small set of parameters as a very useful proxy for a full description of the setting and motion of a SLAM application. We call these motion and structure (MS) parameters, and define them based on information divergence metric. Fig. 21 demonstrates the set of all design spaces.

MS-DSE presents a comprehensive parameterization of 3-D understanding scene algorithms, and thus based on this new parameterization, many new concepts and applications can be developed. One of these applications, active SLAM, is outlined here. For more applications, please see [44] and [105]–[107].

a) Active SLAM: Active SLAM is the method for choosing the optimal camera trajectory, in order to maximize the camera pose estimation, the accuracy of the reconstruction, or the coverage of the environment. In [44], it is shown that MS-DSE can be utilized to optimize not only fixed system parameters, but also to guide a robotic platform to maintain a good performance for localization and mapping. As shown in Fig. 21, a Pareto front holds all optimal parameters. The front has been prepared in advance by exploring the set of all parameters. When the system is operating, optimal parameters are chosen given the desired performance metrics. Then these parameters are used to initialize the system. Using MS parameters, the

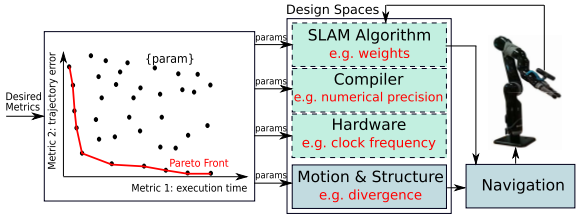

Fig. 21. ***Motion and structure aware active SLAM design space exploration using HyperMapper.***

objective is to avoid motions that cause very high statistical divergence between two consecutive frames. This way, we can provide a robust SLAM algorithm by allowing the tracking work all the time. Fig. 22 compares the active SLAM with a random walk algorithm. The experiments were done in four different environments. In each environment, each algorithm was run ten times. Repeated experiments serve as a measure of the robustness of the algorithm in dealing with uncertainties rising from minor changes in illumination, or inaccuracies of the response of the controller or actuator to the commands. The consistency of the generated map was evaluated manually as either a success or failure of SLAM. If duplicates of one object were present in the map, it was considered as failure. This experiment shows more than 50% success rate in SLAM when employing the proposed active SLAM algorithm [44], an improvement in the *robustness* of SLAM algorithms by relying on design space exploration.

3) Comparative DSE of Dense Versus Semi-Dense SLAM: Another different direction in any DSE work is the performance exploration across multiple algorithms. While multidomain DSE explores different parameters of a given algorithm, the comparative DSE, presented in [108], explores the performance of two different algorithms under different parametric choices.

In comparative DSE, two state-of-the-art SLAM algorithms, KinectFusion and LSD-SLAM, are compared on multiple data sets. Using SLAMBench benchmarking capabilities, a full design space exploration is performed over algorithmic parameters, compilation flags and multiple architectures. Such thorough parameter space exploration gives us key insights on the behavior of each algorithm in different operative conditions and the relationship between different sets of distinct, yet correlated, parameters blocks.

As an example, in Fig. 23 we show the result of comparative DSE between LSD-SLAM and KinectFusion in terms

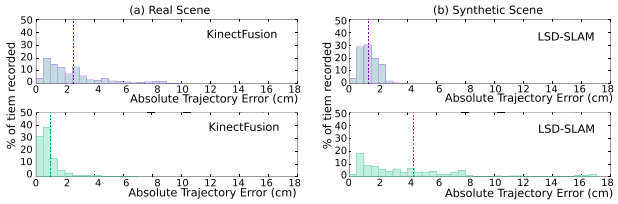

Fig. 23. ***Distribution of absolute trajectory error (ATE) using KinectFusion and LSD-SLAM, run with default parameters on Desktop. The mean absolute error has been highlighted: (a) TUM RGB-D fr2_xyz and (b) ICL-NUIM lr_kt2.***

of their ATE distribution across two scenes of two different data sets. The histograms display the error distribution across the entire sequence, from which we can get a sense of how well the algorithms are performing for the *whole* trajectory. We hope that these analyses enable researchers to develop more robust algorithms. Without the holistic approach enabled by SLAMBench such insights would have been much harder to obtain. This sort of information is invaluable for a wild range of SLAM practitioners, from VR/AR designers to roboticists that want to select/modify the best algorithm for their particular use case.

B. Crowdsourcing

The SLAMBench framework and more specifically its various KinectFusion implementations has been ported to Android. More than 2000 downloads have been made since its official release on the Google Play store. We received numerous positive feedback reports and this application has generated a great deal of interest in the community and with industrial partners.

This level of uptake allowed us to collect data from more than 100 different mobile phones. Fig. 24 shows the speedup across many models of Android devices that we have experimented with. Clearly, it is possible to achieve more than twice *runtime speed* by tuning the system parameters using the tools introduced in the paper. We plan to use these data to analyze the performance of KinectFusion on those platforms and to provide techniques to optimize KinectFusion performance depending of the targeted platform. This work will apply transfer-learning methodology. We believe that by combining design space exploration [106] and the collected data, we can train a decision machine to select code variants and configurations for diverse mobile platforms automatically.

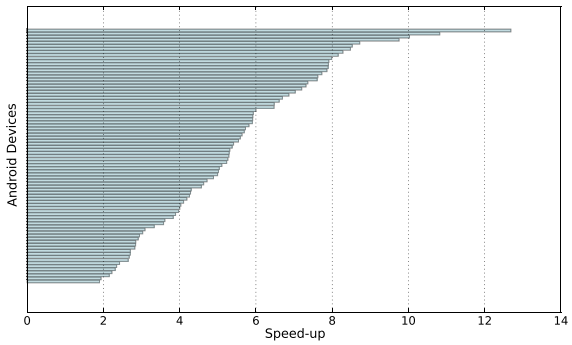

Fig. 24. ***By combining design space exploration and crowdsourcing, we checked that design space exploration efficiently works on various types of platforms. This figure demonstrates the speed-up of the KinectFusion algorithm on various different types of Android devices. Each bar represents the speed-up for one type (model) of Android device. The models are not shown for the sake of clarity of the figure.***

VI. CONCLUSION

In this paper we focused on SLAM, which is an enabling technology in many fields including virtual reality, augmented reality, and robotics. The paper presented several contributions across hardware architecture, compiler and runtime software systems, and computer vision algorithmic layers of SLAM pipeline. We proposed not only contributions at each layer, but also holistic methods that optimize the system as a whole.

In computer vision and applications, we presented benchmarking tools that allow us to select a proper data set and use it to evaluate different SLAM algorithms. SLAMBench is used to evaluate the KinectFusion algorithm on various different hardware platforms. SLAMBench2 is used to compare various SLAM algorithms very efficiently. We also extended the KinectFusion algorithm, such that it can be used in robotic path planning and navigation algorithms by mapping both occupied and free space of the environment. Moreover, we explored new sensing technologies such as focal-plane sensor–processor arrays, which have low power consumption and high effective frame rate.

The software layer of this project demonstrated that software optimization can be used to deliver significant improvements in power consumption and speed tradeoff when specialized for computer vision applications. We explored *static*, *dynamic*, and *hybrid* approaches and focused their application on the KinectFusion algorithm. Being able to select and deploy optimizations adaptively is particularly beneficial in the context of dynamic runtime environment where application-specific details can strongly improve the result of JIT compilation and thus the speed of the program.

The project has made a range of contributions across the hardware design and development field. Profiling tools have been developed in order to locate and evaluate performance bottlenecks in both native and managed applications. These bottlenecks could then be addressed by a range of specialization techniques, and the specialized hardware evaluated using the presented simulation techniques. This represents a full workflow for creating new hardware for computer vision applications which might be used in future platforms.

Finally, we report on holistic methods that exploit our ability to explore the design space at every level in a holistic fashion. We demonstrated several design space exploration methods where we showed that it is possible to fine-tune the system such that we can meet desired

performance metrics. It is also shown that we can increase public engagement in accelerating the design space exploration by crowdsourcing.

In future work, two main directions will be followed: The first is exploiting our knowledge from all domains of this paper to select a SLAM algorithm and design a chip that is customized to efficiently implement the algorithm. This approach will utilize data from SLAMBench2 and real-world experiments to drive the design of a specialized vision processor. The second direction is utilizing the tools and techniques presented here to develop a standardized method that takes the high-level scene understanding functionalities and develops the optimal code that maps the functionalities to the heterogeneous resources available, optimizing for the desired performance metrics. ■

REFERENCES

[1] C. Cadena *et al.*, "Past, present, and future of simultaneous localization and mapping: Toward the robust-perception age," *IEEE Trans. Robot.*, vol. 32, no. 6, pp. 1309–1332, Dec. 2016.

[2] S. Thrun, W. Burgard, and D. Fox, *Probabilistic Robotics (Intelligent Robotics and Autonomous Agents)*. Cambridge, MA, USA: MIT Press, 2005.

[3] H. Durrant-Whyte and T. Bailey, "Simultaneous localization and mapping: Part I," *IEEE Robot. Autom. Mag.*, vol. 13, no. 2, pp. 99–110, Jun. 2006.

[4] A. J. Davison, I. D. Reid, N. D. Molton, and O. Stasse, "MonoSLAM: Real-time single camera SLAM," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 29, no. 6, pp. 1052–1067, Jun. 2007.

[5] G. Klein and D. Murray, "Parallel tracking and mapping on a camera phone," in *Proc. IEEE ACM Int. Symp. Mixed Augmented Reality (ISMAR)*, Oct. 2009, pp. 83–86.

[6] R. A. Newcombe, S. J. Lovegrove, and A. J. Davison, "DTAM: Dense tracking and mapping in real-time," in *Proc. Int. Conf. Comput. Vis. (ICCV)*, 2011, pp. 2320–2327.

[7] R. A. Newcombe *et al.*, "KinectFusion: Real-time dense surface mapping and tracking," in *Proc. IEEE Int. Symp. Mixed Augmented Reality (ISMAR)*, Oct. 2011, pp. 127–136.

[8] L. Fan, F. Zhang, G. Wang, and Z. Liu, "An effective approximation algorithm for the malleable parallel task scheduling problem," *J. Parallel Distrib. Comput.*, vol. 72, no. 5, pp. 693–704, 2012.

[9] N. Melot, C. Kessler, J. Keller, and P. Eitschberger, "Fast crown scheduling heuristics for energy-efficient mapping and scaling of moldable streaming tasks on manycore systems," *ACM Trans. Architecture Code Optim.*, vol. 11, no. 4, pp. 62–1 2015.

[10] N. Melot, C. Kessler, and J. Keller, "Improving energy-efficiency of static schedules by core consolidation and switching off unused cores," in *Proc. Int. Conf. Parallel Comput. (ParCo)*, 2015, pp. 285–294.

[11] H. Xu, F. Kong, and Q. Deng, "Energy minimizing for parallel real-time tasks based on level-packing," in *Proc. IEEE Int. Conf. Embedded Real-Time Comput. Syst. Appl. (RTCSA)*, Aug. 2012, pp. 98–103.

[12] T. Schwarzer, J. Falk, M. Glaß, J. Teich, C. Zebelein, and C. Haubelt, "Throughput-optimizing compilation of dataflow applications for multi-cores using quasi-static scheduling," in *Proc. ACM Int. Workshop Softw. Compilers Embedded Syst.*, 2015, pp. 68–75.

[13] U. Dastgeer and C. Kessler, "Performance-aware composition framework for GPU-based systems," *J. Supercomput.*, vol. 71, no. 12, pp. 4646–4662, 2015.

[14] U. Dastgeer and C. Kessler, "Smart containers and skeleton programming for GPU-based systems," *Int. J. Parallel Program.*, vol. 44, no. 3, pp. 506–530, 2016.

[15] I. Böhm, T. J. E. v. Koch, S. C. Kyle, B. Franke, and N. Topham, "Generalized just-in-time trace compilation using a parallel task farm in a dynamic binary translator," ACM Special Interest Group Program. Notices, Lang. vol. 46, no. 6, pp. 74–85, 2011.

[16] K. D. Cooper *et al.*, "Adaptive compilation made efficient,"*ACM Special Interest Group Program. Lang. Notices*, vol. 40, no. 7, pp. 69–77, 2005.

[17] G. Fursin *et al.*, "Milepost GCC: Machine learning enabled self-tuning compiler," *Int. J. Parallel Program.*, vol. 39, no. 3, pp. 296–327, 2011.

[18] Q. Wang, S. Kulkarni, J. Cavazos, and M. Spear, "A transactional memory with automatic performance tuning," *ACM Trans. Archit. Code Optim.*, vol. 8, no. 4, p. 54, 2012.

[19] S. Kulkarni and J. Cavazos, "Mitigating the compiler optimization phase-ordering problem using machine learning," ACM Special Interest Group Program. Notices, Lang. vol. 47, no. 10, pp. 147–162, 2012.

[20] H. Leather, E. Bonilla, and M. F. P. O'Boyle, "Automatic feature generation for machine learning based optimizing compilation," in *Proc. Annu. IEEE/ACM Int. Symp. Code Gener. Optim.*, Mar. 2009, pp. 81–91.

[21] G. Tournavitis, Z. Wang, B. Franke, and M. F. O'Boyle, "Towards a holistic approach to auto-parallelization: Integrating profile-driven parallelism detection and machine-learning based mapping," in *Proc. ACM SIGPLAN Conf. Programm. Design Implement.*, 2009, pp. 177–187.

[22] M. Zuluaga, E. Bonilla, and N. Topham, "Predicting best design trade-offs: A case study in processor customization," in *Proc. Des. Automat. Test Eur. Conf. Exhib. (DATE)*, 2012, pp. 1030–1035.

[23] I. Bohm, B. Franke, and N. Topham, "Cycle-accurate performance modelling in an ultra-fast just-in-time dynamic binary translation instruction set simulator," in *Proc. Int. Conf. Embedded Comput. Syst., Archit.*, Modeling, Simulation, 2010, pp. 1–10.

[24] K. T. Sundararajan, V. Porpodas, T. M. Jones, N. P. Topham, and B. Franke, "Cooperative partitioning: Energy-efficient cache partitioning for high-performance CMPs," in *Proc. IEEE Int. Symp. High-Perform. Comput. Archit.*, Feb. 2012, pp. 1–12.

[25] O. Almer, N. Topham, and B. Franke, "A learning-based approach to the automated design of MPSoC networks," in *Proc. Int. Conf. Archit. Comput. Syst.*, 2011, pp. 243–258.

[26] L. Nardi, "Introducing SLAMBench, a performance and accuracy benchmarking methodology for SLAM," in *Proc. IEEE Int. Conf. Robot. Automat. (ICRA)*, May 2015, pp. 5783–5790.

[27] G. Reitmayr and H. Seichter. *KFusion GitHub*. [Online]. Available: https://github.com/GerhardR/kfusion

[28] A. Handa, T. Whelan, J. McDonald, and A. Davison, "A benchmark for RGB-D visual odometry, 3-D reconstruction and SLAM," in *Proc. IEEE Int. Conf. Robot. Automat. (ICRA)*, May 2014, pp. 1524–1531.

[29] P. Keir, "DAGR: A DSL for legacy OpenCL codes," in *Proc. 1st SYCL Program. Workshop*, 2016.

[30] R. Baghdadi, "PENCIL: A platform-neutral compute intermediate language for accelerator programming," in *Proc. IEEE Int. Conf. Parallel Archit. Compilation (PACT)*, Oct. 2015, pp. 138–149.

[31] *CARP White-Project. PENCIL-SLAMBench GitHub*. [Online]. Available: https://github.com/carpproject/slambench

[32] B. Bodin *et al.*, "SLAMBench2: Multi-objective head-to-head benchmarking for visual SLAM," in *Proc. IEEE Int. Conf. Robot. Automat. (ICRA)*, 2018, pp. 3637–3644.

[33] T. Whelan, S. Leutenegger, R. F. Salas-Moreno, B. Glocker, and A. J. Davison, "ElasticFusion: Dense SLAM without a pose graph," in *Proc. RSS*, 2015.

[34] O. Kähler, V. A. Prisacariu, C. Y. Ren, X. Sun, P. Torr, and D. W. Murray, "Very high frame rate volumetric integration of depth images on mobile device," *IEEE Trans. Vis. Comput. Graphics*, vol. 21, no. 11, pp. 1241–1250, Nov. 2015.

[35] J. Engel, T. Schöps, and D. Cremers, "LSD-SLAM: Large-scale direct monocular SLAM," in *Proc. Eur. Conf. Comput. Vis. (ECCV)*. 2014, pp. 834–849.

[36] R. Mur-Artal and J. D. Tardos, "ORB-SLAM2: An open-source SLAM system for monocular, stereo, and RGB-D cameras," *IEEE Trans. Robot.*, vol. 33, no. 5, pp. 1255–1262, Oct. 2017.

[37] A. J. Davison, I. D. Reid, N. D. Molton, and O. Stasse, "MonoSLAM: Real-time single camera SLAM," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 29, no. 6, pp. 1052–1067, Jun. 2007.

[38] S. Leutenegger, S. Lynen, M. Bosse, R. Siegwart, and P. Furgale, "Keyframe-based visual-inertial odometry using nonlinear optimization," *Int. J. Robot. Res.*, vol. 34, no. 3, pp. 314–334, 2015.

[39] C. Forster, M. Pizzoli, and D. Scaramuzza, "SVO: Fast semi-direct monocular visual odometry," in *Proc. IEEE Int. Conf. Robot. Automat. (ICRA)*, May/Jun. 2014, pp. 15–22.

[40] M. Abouzahir, A. Elouardi, R. Latif, S. Bouaziz, and A. Tajer, "Embedding SLAM algorithms: Has it come of age?" *Robot. Auton. Syst.*, vol. 100, pp. 14–26, Feb. 2018.

[41] D. Jeffrey and S. Davide, "A benchmark comparison of monocular visual-inertial odometry algorithms for flying robot," in *Proc. IEEE Int. Conf. Robot. Automat. (ICRA)*, 2018, pp. 2502–2509.

[42] W. Li *et al.*, "InteriorNet: Mega-scale multi-sensor photo-realistic indoor scenes dataset," in *Proc. Brit. Mach. Vis. Conf. (BMVC)*, 2018.

[43] S. Saeedi, W. Li, D. Tzoumanikas, S. Leutenegger, P. H. J. Kelly, and A. J. Davison. (2018).*Characterising Localization and Mapping Datasets*. [Online]. Available: http://wbli.me/lmdata/

[44] S. Saeedi, L. Nardi, E. Johns, B. Bodin, P. Kelly, and A. Davison, "Application-oriented design space exploration for SLAM algorithms," in *Proc. IEEE Int. Conf. Robot. Automat. (ICRA)*, May/Jun. 2017, pp. 5716–5723.

[45] C. Loop, Q. Cai, S. Orts-Escolano, and P. A. Chou, "A closed-form Bayesian fusion equation using occupancy probabilities," in *Proc. IEEE Int. Conf. 3-D Vis. (3-DV)*, Oct. 2016, pp. 380–388.

[46] E. Vespa, N. Nikolov, M. Grimm, L. Nardi, P. H. J. Kelly, and S. Leutenegger, "Efficient octree-based volumetric SLAM supporting signed-distance and occupancy mapping," *IEEE Robot. Automat. Lett.*, vol. 3, no. 2, pp. 1144–1151, Apr. 2018.

[47] J. D. Gammell, S. S. Srinivasa, and T. D. Barfoot, "Informed RRT ∗: Optimal sampling-based path planning focused via direct sampling of an admissible ellipsoidal heuristic," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst. (IROS)*, Sep. 2014, pp. 2997–3004.

[48] *Point. White-Grey. Bumblebee2 Datasheet*. [Online]. Available: https://www.ptgrey.com/support/downloads/10132

[49] P. Fankhauser, M. Bloesch, D. Rodriguez, R. Kaestner, M. Hutter, and R. Siegwart, "Kinect v2 for mobile robot navigation: Evaluation and modeling," in *Proc. IEEE Int. Conf. Adv. Robot. (ICAR)*, Jul. 2015, pp. 388–394.

[50] H. Kim, A. Handa, R. Benosman, S.-H. Ieng, and A. Davison, "Simultaneous mosaicing and tracking with an event camera," in *Proc. Brit. Mach. Vis. Conf. (BMVC)*. *BMVA Press*, 2014, pp. 566–576.

[51] P. Bardow, A. J. Davison, and S. Leutenegger, "Simultaneous optical flow and intensity estimation from an event camera," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2016, pp. 884–892.

[52] A. Censi and D. Scaramuzza, "Low-latency event-based visual odometry," in *Proc. IEEE Int. Conf. Robot. Automat. (ICRA)*, May 2014, pp. 703–710.

[53] E. Mueggler, B. Huber, and D. Scaramuzza, "Event-based, 6-DOF pose tracking for high-speed maneuvers," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst. (IROS)*, Sep. 2014, pp. 2761–2768.

[54] H. Kim, S. Leutenegger, and A. J. Davison, "Real-time 3-D reconstruction and 6-DoF tracking with an event camera," in *Proc. Eur. Conf. Comput. Vis. (ECCV)*. 2016, pp. 349–364.

[55] R. Dominguez-Castro *et al.*, "A 0.8- μ m CMOS two-dimensional programmable mixed-signal focal-plane array processor with on-chip binary imaging and instructions storage," *IEEE J. Solid-State Circuits*, vol. 32, no. 7, pp. 1013–1026, Jul. 1997.

[56] G. Linan, S. Espejo, R. Dominguez-Castro, and A. Rodriguez-Vazquez, "Architectural and basic circuit considerations for a flexible 128×128 mixed-signal SIMD vision chip," *Analog Integr. Circuits Signal Process.*, vol. 33, no. 2, pp. 179–190, 2002.

[57] J. Poikonen, M. Laiho, and A. Paasio, "MIPA4k: A 64×64 cell mixed-mode image processor array," in *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS)*, May 2009, pp. 1927–1930.

[58] P. Dudek and P. J. Hicks, "A general-purpose processor-per-pixel analog SIMD vision chip," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 52, no. 1, pp. 13–20, Jan. 2005.

[59] P. Dudek, "Implementation of SIMD vision chip with 128×128 array of analogue processing elements," in *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS)*, May 2005, pp. 5806–5809.

[60] S. J. Carey, A. Lopich, D. R. Barr, B. Wang, and P. Dudek, "A 100,000 FPS vision sensor with embedded 535 GOPS/W 256×256 SIMD processor array," in *Proc. IEEE Symp. VLSI Circuits (VLSIC)*, Jun. 2013, pp. C182–C183.

[61] W. Zhang, Q. Fu, and N. J. Wu, "A programmable vision chip based on multiple levels of parallel processors," *IEEE J. Solid-State Circuits*, vol. 46, no. 9, pp. 2132–2147, Sep. 2011.

[62] J. N. P. Martel, L. K. Müller, S. J. Carey, and P. Dudek, "Parallel HDR tone mapping and auto-focus on a cellular processor array vision chip," in *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS)*, May 2016, pp. 1430–1433.

[63] L. Bose, J. Chen, S. J. Carey, P. Dudek, and W. Mayol-Cuevas, "Visual odometry for pixel processor arrays," in *Proc. IEEE Int. Conf. Comput. Vis. (ICCV)*, Oct. 2017, pp. 4614–4622.

[64] P. Viola and M. Jones, "Robust real-time object detection," *Int. J. Comput. Vis.*, vol. 57, no. 2, pp. 137–154, 2002.

[65] T. Debrunner, S. Saeedi, and P. H. J. Kelly, "Automatic kernel code generation for cellular processor arrays," *ACM Trans. Archit. Code Optim.*, 2018.

[66] *Intel. White-Movidius. Intel Movidius Myriad VPU*. [Online]. Available: https://www.movidius.com/myriad2

[67] N. P. Jouppi *et al.*, "In-datacenter performance analysis of a tensor processing unit," in *Proc. ACM Int. Symp. Comput. Archit. (ISCA)*, 2017, pp. 1–12.

[68] P. Ginsbach, T. Remmelg, M. Steuwer, B. Bodin, C. Dubach, and M. F. P. O'Boyle, "Automatic matching of legacy code to heterogeneous APIs: An idiomatic approach," in *Proc. ACM Int. Conf. Archit. Support Program*. 2018, pp. 139–153.

[69] B. Bodin, L. Nardi, P. H. J. Kelly, and M. F. P. O'Boyle, "Diplomat: Mapping of multi-kernel applications using a static dataflow abstraction," in *Proc. IEEE Int. Symp. Modeling, Anal. Simulation Comput. Telecommun. Syst. (MASCOTS)*, Sep. 2016, pp. 241–250.

[70] B. Bodin, A. Munier-Kordon, and B. D. d. Dinechin, "Optimal and fast throughput evaluation of CSDF," in *Proc. ACM Annu. Des. Automat. Conf. (DAC)*, 2016, pp. 160–1.

[71] C. Kotselidis, J. Clarkson, A. Rodchenko, A. Nisbet, J. Mawer, and M. Luján, "Heterogeneous managed runtime systems: A computer vision case study," in *Proc. ACM SIGPLAN/SIGOPS Int. Conf. Virtual Execution Environ.*, 2017, pp. 74–82.

[72] J. Clarkson, C. Kotselidis, G. Brown, and M. Luján, "Boosting Java performance using GPGPUs," in *Proc. Int. Conf. Archit. Comput. Syst. (ARCS)*. 2017, pp. 59–70.

[73] J. Clarkson, J. Fumero, M. Papadimitriou, M. Xekalaki, and C. Kotselidis, "Towards practical heterogeneous virtual machines," in *Proc. ACM MoreVMs Workshop Mod*. 2018, pp. 46–48.

[74] *Beehive Lab. Maxine/MaxSim*. [Online]. Available: https://github.com/beehive-lab

[75] C. Wimmer, M. Haupt, M. L. V. D. Vanter, M. Jordan, L. Daynès, and D. Simon, "Maxine: An approachable virtual machine for, and in, Java," *ACM Trans. Archit. Code Optim.*, vol. 9, no. 4, pp. 30–1 2013.

[76] F. S. Zakkak, M. Haupt, M. L. V. d. Vanter, M. J. Jordan, L. Daynès, and D. Simon, "On the future of research VMs: A hardware/software perspective," in *Proc. ACM MoreVMs Workshop Mod*. 2018, pp. 51–53.

[77] K. Chandramohan and M. F. O'Boyle, "Partitioning data-parallel programs for heterogeneous MPSoCs: Time and energy design space exploration," in *Proc. ACM SIGPLAN/SIGBED Conf*. 2014, pp. 73–82.

[78] K. Chandramohan and M. F. P. O'Boyle, "A compiler framework for automatically mapping data parallel programs to heterogeneous MPSoCs," in *Proc. ACM Int. Conf. Compilers, Archit. Synth. Embedded Syst. (CASE)*, 2014, pp. 9–1.

[79] T. Spink, H. Wagstaff, B. Franke, and N. Topham, "Efficient code generation in a region-based dynamic binary translator," in *Proc. ACM SIGPLAN/SIGBED Conf*. 2014, pp. 3–12.

[80] H. Wagstaff, M. Gould, B. Franke, and N. Topham, "Early partial evaluation in a JIT-compiled, retargetable instruction set simulator generated from a high-level architecture description," in *Proc. ACM Annu. Des. Automat. Conf. (DAC)*, 2013, pp. 21–1.

[81] H. Wagstaff, T. Spink, and B. Franke, "Automated ISA branch coverage analysis and test case generation for retargetable instruction set simulators," in *Proc. IEEE Int. Conf. Compilers, Archit. Synth. Embedded Syst. (CASES)*, Oct. 2014, pp. 1–10.

[82] T. Spink, H. Wagstaff, B. Franke, and N. Topham, "Efficient dual-ISA support in a retargetable, asynchronous dynamic binary translator," in *Proc. IEEE Int. Conf. Embedded Comput. Syst., Archit., Modeling Simulation (SAMOS)*, Jul. 2015, pp. 103–112.

[83] H. Wagstaff and T. Spink. *The GenSim ADL Toolset*. [Online]. Available: http://www.gensim.org/

[84] K. Kaszyk, H. Wagstaff, T. Spink, B. Franke, M. F. P. O'Boyle, and H. Uhrenholt, "Accurate emulation of a state-of-the-art mobile CPU/GPU platform," in *Proc. Design Automat. Conf. (DAC)*, 2018.

[85] T. Spink, H. Wagstaff, and B. Franke, "Efficient asynchronous interrupt handling in a full-system instruction set simulator," *ACM SIGPLAN Notices*, vol. 51, no. 5, pp. 1–10, 2016.

[86] J. Mawer, O. Palomar, C. Gorgovan, A. Nisbet, W. Toms, and M. Luján, "The potential of dynamic binary modification and CPU-FPGA SoCs for simulation," in *Proc. IEEE Annu. Int. Symp. Field-Programm. Custom Comput. Mach. (FCCM)*, Apr./May 2017, pp. 144–151.

[87] C. Gorgovan, A. D'Antras, and M. Luján, "MAMBO: A low-overhead dynamic binary modification tool for ARM," *ACM Trans. Archit. Code Optim.*, vol. 13, no. 1, pp. 14–1 2016.

[88] C. Gorgovan. *MAMBO: A Low-Overhead Dynamic Binary Modification Tool for ARM*. [Online]. Available: https://github.com/beehive-lab

[89] D. L. Bruening, "Efficient, transparent, and comprehensive runtime code manipulation," Ph.D. dissertation, *Massachusetts Inst. Technol.*, Cambridge, MA, USA, 2004.

[90] A. Rodchenko, C. Kotselidis, A. Nisbet, A. Pop, and M. Luján, "MaxSim: A simulation platform for managed applications," in *Proc. IEEE Int. Symp. Perform. Anal. Syst. Softw. (ISPASS)*, Apr. 2017, pp. 141–152.

[91] D. Sanchez and C. Kozyrakis, "ZSim: Fast and accurate microarchitectural simulation of thousand-core systems," in *Proc. ACM Annu. Int. Symp. Comput. Archit. (ISCA)*, 2013, pp. 475–486.

[92] S. Li, J. H. Ahn, R. D. Strong, J. B. Brockman, D. M. Tullsen, and N. P. Jouppi, "McPAT: An integrated power, area, and timing modeling framework for multicore and manycore architectures," in *Proc. IEEE/ACM Int. Symp. Microarchitecture (MICRO)*, Dec. 2009, pp. 469–480.

[93] A. Rodchenko, C. Kotselidis, A. Nisbet, A. Pop, and M. Luján, "Type information elimination from objects on architectures with tagged pointers support," *IEEE Trans. Comput.*, vol. 67, no. 1, pp. 130–143, Jan. 2018.

[94] V. Sze, "Designing hardware for machine learning: The important role played by circuit designers," *IEEE Solid State Circuits Mag.*, vol. 9, no. 4, pp. 46–54, Sep. 2017.

[95] *Graphcore*. [Online]. Available: https://www.graphcore.ai/

[96] A. Eliazar and R. Parr, "DP-SLAM: Fast, robust simultaneous localization and mapping without predetermined landmarks," in *Proc. Int. Joint Conf. Artif. (IJCAI)*. San Mateo, CA, USA: 2003, pp. 1135–1142.

[97] S. Dublish, V. Nagarajan, and N. Topham, "Characterizing memory bottlenecks in GPGPU workloads," in *Proc. IEEE Int. Symp. Workload Characterization (IISWC)*, Sep. 2016, pp. 1–2.

[98] S. Dublish, V. Nagarajan, and N. Topham, "Cooperative caching for GPUs," *ACM Trans. Archit. Code Optim.*, vol. 13, no. 4, pp. 39–1 2016.

[99] E. Tomusk, C. Dubach, and M. F. P. O'Boyle, "Measuring flexibility in single-ISA heterogeneous processors," in *Proc. ACM Int. Conf. Parallel Archit. Compilation*, 2014, pp. 495–496.

[100] E. Tomusk and C. Dubach, "Diversity: A design goal for heterogeneous processors," *IEEE Comput. Archit. Lett.*, vol. 15, no. 2, pp. 81–84, Jul./Dec. 2016.

[101] E. Tomusk, C. Dubach, and M. F. P. O'Boyle, "Four metrics to evaluate heterogeneous multicores," *ACM Trans. Archit. Code Optim.*, vol. 12, no. 4, pp. 37–1 2015.

[102] E. Tomusk, C. Dubach, and M. F. P. O'Boyle, "Selecting heterogeneous cores for diversity," *ACM Trans. Archit. Code Optim.*, vol. 13, no. 4, pp. 49–1 2016.

[103] E. Kang, E. Jackson, and W. Schulte, A. An, *An Approach for Effective Design Space Exploration*. Berlin, Germany: Springer, 2011, pp. 33–54.

[104] Z. Zhang, A. Suleiman, L. Carlone, V. Sze, and S. Karaman, "Visual-inertial odometry on chip: An algorithm-and-hardware co-design approach," *Robot., Sci. Syst.*, 2017.

[105] B. Bodin *et al.*, "Integrating algorithmic parameters into benchmarking and design space exploration in 3-D scene understanding," in *Proc. ACM Int. Conf. Parallel Archit. Compilation (PACT)*, 2016, pp. 57–69.

[106] L. Nardi, B. Bodin, S. Saeedi, E. Vespa, A. J. Davison, and P. H. J. Kelly, "Algorithmic performance-accuracy trade-off in 3-D vision applications using hypermapper," in *Proc. Int. Workshop Autom. Perform. Tuning (iWAPT), IEEE Int. Parallel Distrib. Process. Symp. (IEEE IPDPS)*, May 2017, pp. 1434–1443.

[107] D. Koeplinger *et al.*, "Spatial: A language and compiler for application accelerators," in *Proc. ACM SIGPLAN Conf. Program.* 2018, pp. 296–311.

[108] M. Z. Zia *et al.*, "Comparative design space exploration of dense and semi-dense SLAM," in *Proc. IEEE Int. Conf. Robot. Automat. (ICRA)*, May 2016, pp. 1292–1299.

ABOUT THE AUTHORS

Authors' photographs and biographies not available at the time of publication.