Accelerating Vision and Navigation Applications on a Customizable Platform

Jason Cong, Beayna Grigorian, Glenn Reinman, Marco Vitanza
Department of Computer Science
University of California, Los Angeles
{cong, bgrigori, reinman, mvitanza}@cs.ucla.edu

***Abstract*—The domain of vision and navigation often includes applications for feature tracking as well as simultaneous localization and mapping (SLAM). As these problems require computationally demanding solutions, it is challenging to achieve high performance without sacrificing the fidelity of results or otherwise consuming excessive amounts of energy. Our goal then is to accelerate the applications in this domain to meet real-time performance constraints while simultaneously reducing energy consumption and avoiding degradation in the quality of results. To achieve this domain-specific acceleration, we model a customizable hardware platform based on the 3D integration of a Field-Programmable Gate Array (FPGA) atop a standard chip multiprocessor (CMP) with Through-Silicon Vias (TSVs) used for communication between the two layers. Furthermore, partial automation of accelerator creation using C-to-RTL tools allows for analysis of a wide range of candidates. In this work, we mathematically characterize viable accelerator candidates, describe ideal application code for acceleration, and outline a dynamic-programming-based methodology for selecting an optimal set of candidates. Our results yield an overall speedup and energy reduction of 9.56X along with a 94X EDP reduction for the domain. Finally, we investigate the effects of various interconnect models on our performance improvements. Overall, our proposed system is shown to be highly efficient in both accelerating performance and saving energy for compute-intensive applications in this domain.**

***Index Terms*—vision, navigation, real-time, low-energy, FPGA, accelerators, customizable architecture**

I. Introduction

Within the domain of vision and navigation, applications often involve computationally-intensive data processing to achieve high levels of situational awareness, making it difficult to meet both low-energy and real-time performance constraints. In autonomous robotics, for instance, the system must ensure that intelligent decisions are made to allow a robot to safely explore and interact with an environment. While it is imperative that the robot responds in real-time to its dynamically-changing surroundings, this type of mobile system is likely also limited by the total amount of energy it can consume. This research investigates customizable, FPGA-based acceleration and energy reduction using 3D integration with CMPs for vision and navigation applications, as in autonomous robotics. Our overall goal is to achieve low-energy real-time performance, demonstrated through reductions in Energy-Delay-Product (EDP), for a set of key algorithms within this domain – specifically, feature tracking along with localization and mapping.

Recent work in feature tracking and sequential map-building and localization resorts to limiting the number of features considered or the frequency of updates made in order to achieve real-time performance [1], [2], [3]. Alternatively, we explore performance enhancement without result degradation by using hardware accelerators that speed up existing full-featured application code.

To achieve the performance improvements we seek for our application domain, we turn to high-performance, energy-efficient heterogenous computing. Our proposed system model is a 3D integration of a Field-Programmable Gate Array (FPGA) atop a standard chip multiprocessor (CMP), with Through-Silicon Vias (TSVs) as the communication medium between the two layers (Figure 1) – a more detailed description of the architecture can be found in [4]. Through 3D integration using TSVs, our design provides efficient and direct CMP-FPGA communication. Although this case study is carried out with the assumption of TSV-based communication, we also make comparisons with other potential interconnects to ensure that our acceleration techniques are not limited to the assumed model.

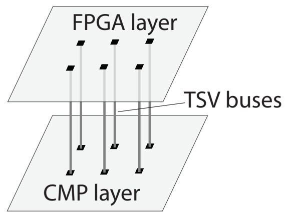

Fig. 1: 3D CMP-FPGA Platform

In this scheme, applications run normally on the CMP, except for key portions that are executed by hardware accelerators implemented on the FPGA. FPGAs are an attractive option because their customizability can lead to large reductions in both computation time and energy usage when compared to software running on a CPU, yet they are relatively low-cost compared to ASIC designs. Furthermore, with the use of the C-to-RTL high-level synthesis tool AutoPilot [5], we are able to quickly create and analyze a wide range of potential accelerators in a semi-automated fashion. With this approach, we model an accelerator platform that can be readily customized to the needs of specific workloads.

The application domain and our representative benchmarks are presented in detail in Section II, followed by an outline of our approach and implementation in Sections III and IV, respectively. We highlight our results in Section V, discuss related work in Section VI, and present our conclusions in Section VII.

II. Application Domain

In computer vision, pattern detection, object recognition, and feature tracking are among the most notable areas [6]. Feature tracking uses various methods for initially detecting features (i.e. points of interest) within a given data set, followed by extracting descriptions of those features [7], [8], [9]. The detected features are subsets of the data, for instance edges or corners in an image, and are described using additional computations, such as surrounding gradients or local histograms. There also exist several methods [10], [11], [12], [13] to solve the simultaneous localization and mapping problem (SLAM) [14]; these entail dynamically collecting data from the environment in order to create a local map that identifies free space and obstacles while simultaneously self-localizing within this map.

In this study, we focus on four specific algorithms to represent the vision and navigation domain: Simultaneous Localization and Mapping (SLAM) based on the Extended Kalman Filter (EKF) [15], [16]; feature detection and description using the Scale-Invariant Feature Transform (SIFT) algorithm [17]; feature detection and description using the Speeded-Up Robust Features (SURF) algorithm [18]; and feature detection using the Kanade-Lucas-Tomasi corner-detection algorithm (KLT) [19] with feature description using the Intensity-Domain Spin Image (IDSI) algorithm [20].

We define the domain using this set of applications for several reasons. First, the EKF is a widely-accepted, standard method for nonlinear state estimation and navigation [16], [21], [22], [23]. Second, feature tracking is a large area of research within computer vision, and includes many widely-used algorithms. Although typically only one would be used by a particular application, by surveying multiple algorithms we intend to identify which ones are more amenable to our method of acceleration. In doing so, we are able to identify algorithmic characteristics that should be sought after in other applications to allow for successful acceleration.

As benchmarks, we use the sample applications `kf-slam` and `features-matching` from the Mobile Robot Programming Toolkit (MRPT) [24], which uses the Open Source Computer Vision (OpenCV) library [25] for the implementations of the SURF and KLT algorithms.

The `kf-slam` application runs an EKF-based SLAM iteration for each action-observation pair in a sample data set. An action-observation pair is a set of data representing a state change by way of a pose transition and an observation of the new state. Since the system is assumed to be non-linear, this data-pair must first be used in a linear approximation based on the previous state prior to applying the Kalman Filter for the next state estimation. A complete iteration of the EKF-based SLAM involves the following steps: acquire the action data and linearize the dynamics of the transition based on the previous filtered state estimate; predict the next state estimate, compute transition Jacobians, and predict the covariance of the next state estimate; acquire the observation data and linearize the dynamics of the observation based on the predicted next state estimate; compute observation Jacobians and predict observation covariances; and finally, using the near-optimal Kalman gain, update the next state estimate and its covariance.

The `features-matching` application has three main stages. First, it runs a feature detection algorithm (SIFT/SURF/KLT) on a sample image to find feature points. Next, it computes descriptors (SIFT/SURF/IDSI) for each feature. Finally, it matches the features to those in a previously-analyzed image by computing the minimum descriptor distance between each newly-found feature and every feature in the previous image. By matching features between successive images, the movement of each feature can be tracked.

With these benchmarks serving as baseline implementations, we aim to create custom accelerators that will allow them to perform in real-time with reduced energy consumption. In an effort to match data collection frequencies of current autonomous systems [26], [27], we have assumed input rates of 30 Hz for image data and 1 Hz for range-and-bearing sensor data. Our measurements indicate that speed-ups of at least 30X for SIFT-based `features-matching`, 9X for SURF-based `features-matching`, 3.6X for KLT-IDSI-based `features-matching`, and 12.4X for `kf-slam` are needed to meet real-time constraints.

III. Approach

Given the area constraints of the FPGA and the limited bandwidth of the off-chip interconnect, our design goals are to find accelerators that achieve enough speedup to push execution past the thresholds set by real-time data-collection frequencies. While our primary focus is on reducing execution time, saving energy also remains a significant enough goal such that we may choose to include accelerators that achieve little speedup so long as they reduce energy consumption.

The creation of effective accelerators begins with thorough application profiling to determine the sets of operations that consume the largest amounts of time (and energy). We measure program run-time statistics for our benchmarks using hardware counters accessed with the PAPI library [28]. Energy consumption is estimated using the McPAT framework [29]. Based on profiling results, we identify accelerator candidates (Section III-A) and synthesize RTL designs using the AutoPilot toolchain to determine latency and resource consumption, allowing us to systematically select the optimal subset of candidates for our domain (Section III-B). Finally, we implement the selected RTL designs for the target FPGA device using Xilinx ISE [30] map, place, and route tools, followed by analysis of latency and energy consumption of the completed accelerators using Xilinx XPower Analyzer [31].

A. Accelerator Candidate Identification

In order to characterize accelerator candidates based on the profiling results, we introduce the terms functional speedup (FS), application speedup (AS), and domain speedup (DS). We define FS as the speedup of a particular function using

TABLE I: Time and energy statistics for `kf-slam` and `features-matching` benchmarks

	Cycles (at 2 GHz)	**Energy (J)**	AS_{max}	AER_{max}	DS_{max}	DER_{max}
`kf-slam` (EKF-SLAM)	69,943,334,047	342.43	–	–	–	–
`jacobiansPoseComposition`	43,768,831,168	211.89	2.64	2.62	2.44	2.42
`sphericalCoordinates`	21,911,609,872	106.73	1.46	1.45	1.42	1.42
`features-matching` (SIFT)	2,399,480,515	12.12	–	–	–	–
`SIFTextremum`	253,435,933	1.26	1.12	1.12	1.00	1.00
`SIFTdesc`	828,914,977	4.15	1.53	1.52	1.01	1.01
`features-matching` (SURF)	782,557,518	4.09	–	–	–	–
`findMaximaInLayer`	9,352,586	0.05	1.01	1.01	1.00	1.00
`SURFdesc`	378,550,056	1.99	1.94	1.95	1.01	1.01
`features-matching` (KLT-IDSI)	478,421,485	2.44	–	–	–	–
`cornerMinEigenvals`	26,038,388	0.14	1.06	1.06	1.00	1.00
`computeIDSI`	393,453,604	2.00	5.63	5.54	1.01	1.01

an FPGA-based accelerator, AS as the speedup of the entire application due to one or more accelerators, and DS as the speedup of the entire domain of applications due to a selected set of accelerators. Hence, for an accelerator that transforms a function from T cycles to T' cycles, the functional speedup is simply expressed as:

$$FS = \frac{T}{T'}$$

Consider an application that takes a total of T_A cycles and has a set of m accelerators that achieve speedups of FS_1, …, FS_m for functions that originally take T_1, …, T_m cycles and are run R_1, …, R_m times, respectively; we then express the speedup of that application due to all the accelerators as follows:

$$AS = \frac{T_A}{(T_A - R_1T_1 - \ldots - R_mT_m) + R_1\frac{T_1}{FS_1} + \ldots + R_m\frac{T_m}{FS_m}}$$

For a domain of n applications having T_{A_1}, …, T_{A_n} total cycles and AS_1, …, AS_n application speedups, respectively, the overall domain speedup becomes:

$$DS = \frac{T_{A_1} + \ldots + T_{A_n}}{\frac{T_{A_1}}{AS_1} + \ldots + \frac{T_{A_n}}{AS_n}}$$

Unlike FS, AS and DS are sensitive to the run-counts of the code being replaced by the accelerator. Note that these speedup metrics also take into account CMP-FPGA communication overhead with respect to the proposed interconnect model (i.e., the T' values include cycles dedicated to communication).

In accordance with Amdahl's Law, a single accelerator α can achieve up to the following amount of application speedup (i.e. $FS \to \infty$):

$$AS_{max} = \frac{T_A}{T_A - R_\alpha \cdot T_\alpha} = \frac{1}{1 - \frac{R_\alpha \cdot T_\alpha}{T_A}} = \frac{1}{1-P}$$

where P is the proportion of the application affected by that particular accelerator (i.e. $0 \leq P \leq 1$). Assuming that the $n-1$ other applications within the domain execute in $T_{A_2}, \ldots, T_{A_n}$ total numbers of cycles and in the worst case they cannot be accelerated by α, the maximum amount of domain speedup this accelerator can achieve is:

$$DS_{max} = \frac{T_A + T_{A_2} + \ldots + T_{A_n}}{T_A \cdot (1-P) + T_{A_2} + \ldots + T_{A_n}}$$

The maximum application and domain energy reductions (i.e. AER_{max} and DER_{max}) are defined similarly.

Table I shows AS_{max}, AER_{max}, DS_{max}, and DER_{max} for each of the accelerator candidates in our benchmark suite. Note that as there is strong correlation between speedup and energy reduction (e.g. AS_{max} is roughly equal to AER_{max}), energy is not explicitly considered for accelerator identification/selection. Descriptions of the accelerator candidates can be found in Section IV.

As Amdahl's law dictates, a candidate must make up a significant portion of an application in order to have a non-negligible effect on overall performance. However, large code sizes lead to bloated accelerators that require excessive amounts of FPGA resources. The process of identifying viable candidates therefore involves balancing the tradeoff between AS_{max} and accelerator code size.

We characterize an effective accelerator candidate as one with a relatively high ratio of AS_{max} to accelerator code size (i.e. resource consumption). At one extreme, the entire application is accelerated, meaning $AS_{max} = \infty$, which could lead to very high FPGA resource consumption (possibly over 100%). At the other extreme, multiple small portions of an application are accelerated, in which case overall FPGA resource consumption is lower and better controlled. However, AS_{max} is also considerably reduced.

Ideal candidates, we find, likely exist in the compute-intensive inner sections of large loops within a given application. In this case, not only is resource consumption low (i.e. the same accelerator is reused during each iteration), but overall acceleration due to functional speedup is found to far outweigh the overhead of repeated FPGA communication. This can be seen by the high speedups for EKF-SLAM (refer to Figure 2), which uses accelerators of this ideal type. Consistent with our characterization of effective candidates, this methodology leads to accelerators with high AS_{max} values and small code sizes, resulting in high ratios of the two.

B. Accelerator Selection

Once the accelerator candidates are identified, we synthesize them in hardware using AutoPilot and select the most suitable accelerators to bundle together on a single FPGA. Being constrained by the size of the FPGA, we are faced with limitations on the number of flip-flops (FFs) and look-up tables (LUTs), which are enclosed into fixed-size logic slices. There are also a limited number of digital signal processing slices (DSPs) and block RAM modules (BRAMs). In order to use these resources optimally, we analyze the domain speedup (DS) of each accelerator and compare this to its overall resource consumption.

We approach this problem of optimal candidate-set selection from a dynamic-programming perspective. Specifically, we reduce this to a multi-dimensional knapsack problem where we maximize cumulative DS while being restricted by the available FPGA resources. This method differs from the traditional knapsack problem in that three sets of weights must be tracked simultaneously – logic slices, DSPs, and BRAMs. Additionally, a direct summation of DS values does not provide a correct representation of the combined speedup. Instead, the following formula must be used to represent the effective speedup of a group of n accelerators with DS values $DS_1, \ldots, DS_n$:

$$DS_{effective} = \frac{1}{\sum_{i=1}^{n}\left(\frac{1}{DS_i}\right) - (n-1)}$$

Despite this added complexity, approximate solutions are achievable in polynomial time [32]. Specifically, for n accelerators and 3 constraints, the time complexity of a solution with maximum error ϵ is roughly $O(n^k)$, where $k = \min(n, \lceil 3(1-\epsilon)/\epsilon \rceil)$.

We note that for the small set of candidates we present in this paper, a brute-force selection algorithm would suffice. However, our dynamic-programming approach will scale well to larger sets of candidates. Using these methods, we select the following accelerators: `jacobiansPoseComposition`, `sphericalCoordinates`, `computeIDSI`, `SURFdesc`, and `SIFTextremum`.

IV. Accelerator Implementation

We implement our accelerators by manually extracting and translating portions of our applications' C++ code into AutoPilot-friendly C. The AutoPilot tool generates a Register-Transfer Level (RTL) description of an algorithm from a C-based functional description. Automatic loop pipelining, unrolling, and/or merging can be applied if desired. AutoPilot supports most C constructs such as functions, arrays, and structures. Pointers are supported, but all pointer values must be statically determinable. This means that all dynamic memory allocations must be removed. Additionally, we reimplement necessary mathematical functions such as $\exp(x)$ and $\mathrm{sqrt}(x)$ in C, as hardware implementations are not automatically generated. Aside from making these modifications, the functionality of the code is kept intact so as to maintain the fidelity of the original applications. The accelerator functionalities are detailed in the following sections.

A. *jacobiansPoseComposition*

A pose is defined as a 3D point representing location combined with a quaternion representing orientation. Given two poses x and u, where x is the robot's pose and u is the pose of a sensor on the robot, this EKF-SLAM accelerator computes the pose composition $f(x, u) = x + u$. It then computes the two Jacobians $\frac{df}{dx}$ and $\frac{df}{du}$.

Since this accelerator contains no loops and consists primarily of floating-point arithmetic, it is easily parallelized by our C-to-RTL tools. In addition to being compute-intensive, this set of operations is repeated for each action-observation data-pair, making this an ideal accelerator.

B. *sphericalCoordinates*

Given a sensor pose and the absolute position of a landmark, this second EKF-SLAM accelerator computes the landmark's position in spherical coordinates with respect to the sensor. Also computed are the derivatives of the spherical coordinates with respect to the landmark point and the sensor pose.

Like `jacobiansPoseComposition`, this code is purely computational with no loops and few conditional expressions. As these computations are also repeated for each data-pair throughout the application, this function has been identified as an ideal accelerator candidate.

C. *SIFTextremum*

To accelerate SIFT-based feature matching, this accelerator consists of the core feature-detection algorithm. Given a point in the Difference of Gaussians (DoG) pyramid, we check whether it is the maximum or minimum of its 26 scale-space neighbors. If so, it is a feature candidate, and we iteratively interpolate it through scale-space to make the feature location more precise.

The extremum check is well-suited for a hardware implementation because all 26 neighbors can be checked in parallel. However, numerous memory accesses are required to continuously obtain the different sets of 26 neighbor values. Furthermore, the interpolation process is still roughly sequential as it is implemented as a loop with several input-dependent stopping conditions.

D. *SIFTdesc*

This accelerator computes the SIFT descriptor for a given feature point by constructing a histogram of gradients in an oriented patch around the feature location. As the computation is similar to that of `SURFdesc` and we did not ultimately use this accelerator, we do not describe it in further detail here. However, it is important to note that this candidate is the most complex in nature. Not only are patch sizes dependent on feature-scales, thereby requiring larger data sets to be read, but each point in the patch is linearly interpolated into the histogram, contributing to multiple bins and resulting in added computation.

E. *findMaximaInLayer*

To accelerate SURF feature detection, this accelerator consists of a critical part of the algorithm. It identifies potential features by finding maxima in a 2D array of determinants. Like SIFT, it also interpolates feature-points through scale-space to increase the accuracy of the feature location. Moreover, the process of finding maxima is also memory-intensive and the interpolation similarly exhibits sequential behavior.

F. *SURFdesc*

With this accelerator, the SURF descriptor is computed for a given feature point. First, it determines feature orientation by sliding a radial window around the feature point and computing the distance-weighted sum of gradient angles in the window. The window orientation with the largest gradient response is saved as the feature orientation. Next, the image gradient is sampled in an oriented rectangular patch around the feature point. Finally, the descriptor is assembled as a histogram of gradients in 16 sub-rectangles within the patch.

This accelerator is the most complex of our selections. We use BRAM to store pre-computed sample points and weights for determining feature orientation. We also pipeline the many loops with our C-to-RTL tools.

G. *cornerMinEigenvals*

The main part of the KLT algorithm is contained in this accelerator. It first computes the gradient of pixel intensities throughout the source image. Using this gradient, it computes at each point the minimum eigenvalue of the structure tensor, which is a symmetric matrix of partial-derivatives of image intensity. Similar to SIFT and SURF feature detection, this algorithm applies relatively simple operations across a large image data set, resulting in memory-intensive rather than compute-intensive behavior.

H. *computeIDSI*

Given a feature location, this accelerator computes the Intensity-Domain Spin Image (IDSI) around the feature. An IDSI is a 2D histogram with the axes being intensity difference and distance from the feature point. Histogram values are exponentially weighted so that values close in intensity or distance are more prominent. The histogram is normalized to the range $[0, 1]$ to provide some invariance to illumination.

This accelerator is much more deterministic than `SURFdesc` since the patch size is statically known and there is no concept of feature orientation. Using AutoPilot, most loops in this implementation are pipelined, while a few simpler ones are unrolled.

V. Performance Evaluation

Our system is evaluated using Red Hat 4.1.2 (x86-64) on a 2 GHz Intel Xeon processor with 8 GB main memory. We implement our accelerators for the Xilinx Virtex 6 FPGA (resources summarized in Table III). The benchmark applications are built from MRPT 0.9.3 and OpenCV 2.2, and the sample images used by `features-matching` are 640×480 pixels.

To model our system's interconnect, we assume a single 128-bit time-multiplexed TSV bus. Since the FPGA clock is 20 times slower than the CMP, this effectively results in 20 individual TSV buses, where 4 are allocated to each of our 5 accelerators. For each effective TSV bus, we assume 1 FPGA cycle and $\frac{1}{2}CV^2$ Joules per access, where $C = 0.8$ femtofarads and $V = 1.2$ volts. We find the energy used by the TSVs to be negligible. We treat every load and store on the FPGA (to external data) as a separate, single-cycle access. As we do not attempt any data packing over the buses, our results are conservative and we would expect to see further performance improvement with more efficient use of buses.

Table II lists the time, energy, and speedup statistics for all accelerator candidates. We note that the number of lines of code roughly correlates with the complexity of an accelerator, but not necessarily its run time. From these results, we see that the accelerators which were not selected (i.e. `SIFTdesc`, `findMaximaInLayer`, and `cornerMinEigenvals`) require the highest numbers of bus accesses. Although some of these accesses can occur in parallel (i.e. multiple buses could be in use simultaneously for a single accelerator), the resulting overhead remains large and significantly limits functional speedup. In comparison, for the accelerator with the highest FS (i.e. `jacobiansPoseComposition` with $FS = 2017$), the number of bus accesses (70) is very low. Since this is close to the total number of FPGA cycles (77), this verifies that the accelerator is made up of highly parallel computations which are almost fully hidden by the external load and store operations.

Table III presents the FPGA resource utilization corresponding to each accelerator candidate. Although our selected accelerators consume almost the entire FPGA (96% slices), only `SURFdesc` uses BRAM. For the memory-intensive feature-detection algorithms, it will likely be useful to modify the original code to stream large portions of the input images to FPGA-local BRAM modules. In this way, the FPGA resources will be better utilized, while memory-access latency will also decrease.

Finally, Figure 2 summarizes performance improvements, including speedup, reduction in energy, and reduction in the Energy Delay Product (EDP) for each application as well as the overall domain. Notably, we achieve a 15X speedup for EKF-SLAM and a 4.4X speedup for KLT-IDSI, resulting in enough performance enhancement to meet our previously established real-time constraints. We see similar reductions in energy, ultimately resulting in a 94X EDP reduction for the entire domain. We further point out that despite seeing virtually no speedup, the SIFT accelerator nevertheless provides enough energy reduction to justify occupying the otherwise unused FPGA resources.

In a single application, one would not likely use more than one type of feature detection. However, with our results we are able to identify which algorithms for feature detection are more amenable to this type of acceleration, and hence should be preferred for the assumed architecture. We find that as feature detection is primarily a search algorithm over a large data set, implementations for the SIFT, SURF, and KLT detectors become highly memory-intensive computations that bear much interconnect communication overhead for marginal functional speedups. Consequently, our approach of partial acceleration using C-to-RTL tools turns out to be less effective than, for instance, a full custom FPGA implementation. EKF-SLAM and IDSI, on the other hand, are compute-intensive, making our approach more successful.

As ideal accelerators are likely found in inner sections of

TABLE II: Time, energy, and speedup statistics for accelerator candidates

	Lines of C Code	FPGA Cycles	# Accesses (Reads & Writes)	# Runs	Total CPU Cycles	Total Energy (J)	FS	AS	DS
`jacobiansPoseComposition*`	186	77	70	13,994	21,550,760	0.02	2017.42	2.64	2.44
`sphericalCoordinates*`	203	104	40	12,750	26,520,000	0.03	826.23	1.46	1.42
`SIFTextremum*`	196	10.6 †	42	1,183,935	251,040,500	0.27	1.01	1.00	1.00
`SIFTdesc`	255	121,323	60,788	1,095	2,656,973,700	2.88	0.31	0.57	0.98
`findMaximaInLayer`	222	6,061,448	8,182,740	8	969,831,680	1.05	0.01	0.45	0.99
`SURFdesc*`	304	13,716	5,452	682	187,086,240	0.20	2.02	1.32	1.00
`cornerMinEigenvals`	153	11,275,054	14,209,497	1	112,750,540	0.12	0.23	0.85	1.00
`computeIDSI*`	119	4,442	1,484	254	22,565,360	0.02	17.44	4.45	1.01

Note: All accelerators run at 100 MHz, except for `cornerMinEigenvals` which runs at 200 MHz.
* Selected accelerator. † Weighted average for two nested conditions; the outer condition executes every time the accelerator is called, but is true only about 0.2% of the time.

TABLE III: FPGA resource utilization for accelerator candidates

	BRAM	DSP	FF	LUT	SLICE
`jacobiansPoseComposition*`	0	128	9282	20781	6983
`sphericalCoordinates*`	0	304	20721	37140	11275
`SIFTextremum*`	0	36	6513	13386	4142
`SIFTdesc`	0	38	10097	18274	5594
`findMaximaInLayer`	0	50	14191	25148	7318
`SURFdesc*`	35	139	19887	39783	11825
`cornerMinEigenvals`	0	36	11300	13528	3870
`computeIDSI*`	0	17	3580	6706	2087
Total Available Resources	832	768	301440	150720	37680
FPGA Utilization (* only)	4.21%	81.25%	19.90%	78.16%	96.37%

* Selected accelerator.

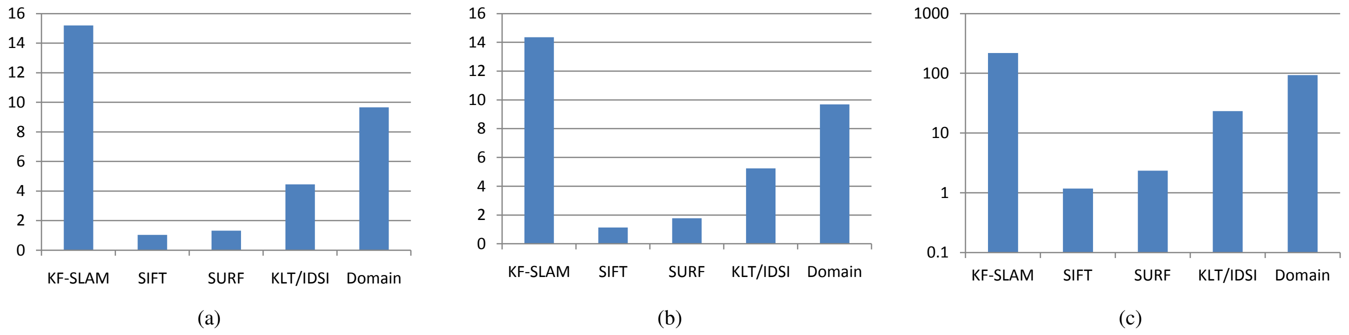

Fig. 2: Time and energy improvements using accelerators: (a) speedup; (b) energy reduction; (c) EDP reduction.

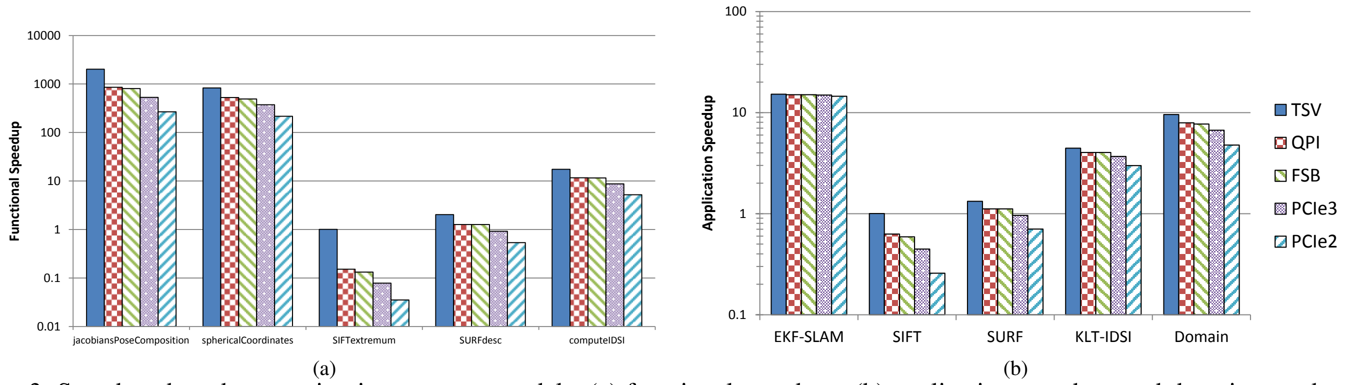

Fig. 3: Speedups based on varying interconnect models: (a) functional speedups; (b) application speedups and domain speedup.

critical loops and are run many times during an application's execution, their relative performance may be highly dependent on communication overhead. For this reason, we evaluate the performance of our accelerators using additional interconnect models. This can also be viewed as a sensitivity study of varying communication bandwidths and latencies.

We model the total execution time of an accelerator with interconnect i using the latency of a memory access L_i and the interconnect's raw 1-way bandwidth BW_i. Since our original measurements account for the assumed 1-cycle 32-bit accesses over the TSV bus, we define the latency penalty LP_i and the bandwidth penalty BP_i to characterize the additional time taken over slower interconnects:

$$LP_i = L_i - L_{TSV} \; ; \; BP_i = \frac{BW_{TSV}}{BW_i} - 1$$

Given an accelerator that originally runs in N cycles of length T_{clk} and has M external memory accesses, our simplified model for the total execution time T_i using interconnect i is then:

$$T_i = LP_i + T_{clk} \cdot (N + M \cdot BP_i)$$

Note that when i is TSV, $T_i = T_{clk} \cdot N$.

We calculate values for FS, AS, and DS for the following interconnects: TSV bus, QuickPath Interconnect (QPI), front-side bus (FSB), PCI-Express 3.0 (PCIe3), and PCI-Express 2.0 (PCIe2). Table IV lists the interconnect model parameters, while Figures 3a and 3b display the corresponding speedup metrics. The latency over QPI is estimated as 2 flits $\times$ 2 cycles/flit $\times$ 3.2 GHz = 1.25 ns [33], the latency over FSB is estimated using manufacturer data [34], and the latency over PCIe is estimated by extrapolating results from [35].

TABLE IV: Interconnect model parameters

	L_i (ns)	BW_i (Gbps)	Description
TSV	0.5	256	128 bits · 2 GHz
QPI	1.25	102.4	20 bits · 64/80 ovhd. · 2 · 3.2 GHz
FSB	105	102.4	64 bits · 4 · 400 MHz
PCIe3	63	64	8 lanes · 1 bit · 8 GHz
PCIe2	126	32	8 lanes · 1 bit · 5 GHz · 4/5 ovhd.

As seen in Figure 3, because the FS values for the EKF-SLAM accelerators are high, the slower interconnects only slightly reduce AS (2% average loss). In contrast, AS for SIFT is drastically reduced (52% average loss) because its large number of initial simultaneous memory accesses make it highly bandwidth-dependent. These impacts on accelerator speedups support the previous evaluations based on bus accesses for accelerators of compute-intensive versus memory-intensive applications. To mitigate increased communication overhead due to decreased interconnect bandwidth, overlap of computation and communication is needed, as done in [36]. However, the functional-speedup reductions seen for our compute-intensive accelerators are not significant enough to affect our accelerator choices.

VI. Related Work

Early efforts in customizable computing [37], [38], [39] were faced with several limitations, including communication bottlenecks, excessive operating system overhead, and restricted programming environments. More recent research, however, has addressed these limitations in the context of developing and integrating hardware-based accelerators. Stillwell et al., for instance, introduce a high-performance accelerator interface for system-on-chips (SoCs) that abstracts the function interface to hardware accelerators and eliminates overhead by including accelerators in the virtual-memory system [40]. This approach is complementary to our work, as our accelerators are essentially C functions with traditional function-call and memory-access semantics. Similarly, recent research by Cong et al. has shown much potential for performance and energy improvement via architectural support in accelerator-rich CMPs [41].

Previous work has also been done to accelerate applications specifically within this domain. Lee and Salcic have implemented an FPGA-based Kalman filtering coprocessor for radar tracking systems [42]. In contrast, we accelerate part of a Kalman filter in the context of the SLAM problem. Moreover, Bonato et al. have implemented a complete SIFT feature detection system on an FPGA [43]. Alternatively, we accelerate SIFT feature detection with minimal effort by directly implementing existing code in hardware. Finally, Lee et al. have implemented custom on-chip accelerators for the SURF feature detection algorithm on a mobile augmented-reality platform [44]. However, we accelerate the SURF descriptor computation using an FPGA-based accelerator.

Finally we note that, to the best of our knowledge, we present the first hardware acceleration of the IDSI descriptor computation.

VII. Conclusion and Future Work

In this paper, we have distinguished a vision and navigation domain composed of the EKF-SLAM, SIFT, SURF, and KLT-IDSI algorithms. We have adopted an FPGA-based model for accelerating this domain of applications while maintaining the original fidelity of results. For the proposed platform, we have characterized effective accelerators and shown how compute-intensive inner sections of loops can be ideal candidates. Furthermore, despite inefficiency in accelerating memory-intensive applications, we have demonstrated that our proposed platform is effective for accelerating compute-intensive ones, particularly for low-energy real-time performance. We have implemented a selected set of hardware accelerators for this domain, and have presented the resulting performance improvements and energy savings. Notably, we achieve speedups of over 15X for SLAM, over 4X for KLT-IDSI-based feature tracking, and over 9.5X for the overall domain, with similar reductions in energy consumption and 94X EDP reduction. Furthermore, even as we adopt alternative interconnect models with lower bandwidth, our compute-intensive accelerators retain significant speedups and remain viable.

Future work in this area includes exploring a more diverse set of applications within this domain and identifying further areas for acceleration. For algorithms that are not a good fit for FPGA acceleration, performance improvement via multi-threading may be a better option. We plan to investigate detailed comparisons between our scheme and multi-threaded benchmarks in order to generate a useful cost/benefit analysis for FPGA-based acceleration. We also plan to explore efficient data packing over the buses to further improve performance and potentially create viable accelerators for memory-intensive applications like SIFT and SURF. Although this would entail hand-written, optimized C or HDL code, it is likely that a library of general image-processing accelerators can be created once and be useful for many different applications. Finally, with the use of performance counters and C-to-RTL tools, it may be possible to automate the entire process of accelerator identification, selection, and implementation.

VIII. Acknowledgments

This research is supported by the Center for Domain-Specific Computing (CDSC), funded by the NSF Expedition in Computing Award CCF-0926127. The authors would also like to thank Yi Zou and Karthik Gururaj for their help in configuring necessary tools and resources.

References

[1] K.-S. Kim, D.-S. Jang, and H.-I. Choi, "Real time face tracking with pyramidal lucas-kanade feature tracker," in *Computational Science and Its Applications - ICCSA 2007*, ser. Lecture Notes in Computer Science, O. Gervasi and M. Gavrilova, Eds. Springer Berlin / Heidelberg, 2007, vol. 4705, pp. 1074–1082.

[2] A. J. Davison and N. Kita, "Sequential localisation and map-building for real-time computer vision and robotics," *Robotics and Autonomous Systems*, vol. 36, no. 4, pp. 171–183, 2001.

[3] J. J. Leonard and H. J. S. Feder, "A computationally efficient method for large-scale concurrent mapping and localization," in *Proceedings of the Ninth International Symposium on Robotics Research*. Springer-Verlag, 1999, pp. 169–176.

[4] J. Cong, K. Gururaj, M. Huang, S. Li, B. Xiao, and Y. Zou, "Domain-specific processor with 3d integration for medical image processing," in *Proceedings of the 22nd IEEE International Conference on Application-specific Systems, Architectures and Processors*, ser. ASAP '11, To Appear.

[5] Z. Zhang, Y. Fan, W. Jiang, G. Han, C. Yang, and J. Cong, *High-Level Synthesis: From Algorithm to Digital Circuit*. Springer Publishers, 2008, ch. 6.

[6] C. H. e. Chen, *Handbook of pattern recognition and computer vision. 4th ed.* Hackensack, NJ: World Scientific., 2010.

[7] H. Zhou, Y. Yuan, and C. Shi, "Object tracking using sift features and mean shift," *Computer Vision and Image Understanding*, vol. 113, no. 3, pp. 345–352, 2009, special Issue on Video Analysis.

[8] S. Birchfield and S. Pundlik, "Joint tracking of features and edges," in *Computer Vision and Pattern Recognition, 2008. CVPR 2008. IEEE Conference on*, June 2008, pp. 1–6.

[9] D. A. Forsyth and J. Ponce, *Computer Vision: A Modern Approach*. Prentice Hall Professional Technical Reference, 2002.

[10] L.-F. Gao, Y.-X. Gai, and S. Fu, "Simultaneous localization and mapping for autonomous mobile robots using binocular stereo vision system," in *Mechatronics and Automation, 2007. ICMA 2007. International Conference on*, Aug 2007, pp. 326–330.

[11] F. Bonin-Font, A. Ortiz, and G. Oliver, "Visual navigation for mobile robots: A survey," *Journal of Intelligent and Robotic Systems*, vol. 53, pp. 263–296, 2008.

[12] K. S. Chong and L. Kleeman, "Sonar based map building for a mobile robot," in *Robotics and Automation, 1997. Proceedings., 1997 IEEE International Conference on*, vol. 2, Apr 1997, pp. 1700–1705.

[13] X. Dai, H. Zhang, and Y. Shi, "Autonomous navigation for wheeled mobile robots-a survey," in *Innovative Computing, Information and Control, 2007. ICICIC '07. Second International Conference on*, Sept 2007, pp. 551–551.

[14] S. Thrun, "Simultaneous localization and mapping," in *Robotics and Cognitive Approaches to Spatial Mapping*, ser. Springer Tracts in Advanced Robotics, M. Jefferies and W.-K. Yeap, Eds. Springer Berlin / Heidelberg, 2008, vol. 38, pp. 13–41.

[15] J.-L. Blanco, "Derivation and implementation of a full 6d ekf-based solution to bearing-range slam," University of Malaga, Spain, http://babel.isa.uma.es/˜jlblanco/papers/RangeBearingSLAM6D.pdf, Technical Report, Mar 2008.

[16] R. E. Kalman, "A new approach to linear filtering and prediction problems," *Journal of Basic Engineering*, vol. 82, no. D, pp. 35–45, 1960.

[17] D. G. Lowe, "Object recognition from local scale-invariant features," *Computer Vision, IEEE International Conference on*, vol. 2, pp. 1150–1157, 1999.

[18] H. Bay, A. Ess, T. Tuytelaars, and L. V. Gool, "Surf: Speeded up robust features," *CVIU*, vol. 110, no. 3, pp. 346–359, 2008.

[19] J. Shi and C. Tomasi, "Good features to track," *Computer Vision and Pattern Recognition, IEEE Computer Society Conference on*, pp. 593–600, 1994.

[20] S. Lazebnik, C. Schmid, and J. Ponce, "A sparse texture representation using local affine regions," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 27, pp. 1265–1278, 2005.

[21] M. Dissanayake, P. Newman, S. Clark, H. Durrant-White, and M. Csorba, "A solution to the simultaneous localization and map building (slam) problem," *Robotics and Automation, IEEE Transactions on*, vol. 17, no. 3, pp. 229–241, 2001.

[22] U. Frese, "A discussion of simultaneous localization and mapping," *Autonomous Robots*, vol. 20, pp. 25–42, 2006.

[23] H. Durrant-Whyte and T. Bailey, "Simultaneous localization and mapping: part i," *Robotics Automation Magazine, IEEE*, vol. 13, no. 2, pp. 99–110, June 2006.

[24] *The Mobile Robot Programming Toolkit*, http://mrpt.org/.

[25] *Open Source Computer Vision*, http://opencv.willowgarage.com/.

[26] A. I. Mourikis, N. Trawny, S. I. Roumeliotis, A. E. Johnson, A. Ansar, and L. Matthies, "Vision-aided inertial navigation for spacecraft entry, descent, and landing," *Trans. Rob.*, vol. 25, no. 2, pp. 264–280, April 2009.

[27] J.-L. Blanco, F.-A. Moreno, and J. Gonzalez, "A collection of outdoor robotic datasets with centimeter-accuracy ground truth," *Autonomous Robots*, vol. 27, no. 4, pp. 327–351, November 2009.

[28] *Performance API*, http://icl.cs.utk.edu/papi/.

[29] S. Li, J. H. Ahn, R. Strong, J. Brockman, D. Tullsen, and N. Jouppi, "Mcpat: An integrated power, area, and timing modeling framework for multicore and manycore architectures," in *Microarchitecture, 2009. MICRO-42. 42nd Annual IEEE/ACM International Symposium on*, Dec 2009, pp. 469–480.

[30] *Xilinx ISE Design Suite*, http://www.xilinx.com/tools/designtools.htm.

[31] *Xilinx XPower Analyzer*, http://www.xilinx.com/products/design_tools/logic_design/verification/xpower_an.htm.

[32] A. M. Frieze and M. R. B. Clarke, "Approximation algorithms for the m-dimensional 0-1 knapsack problem: Worst-case and probabilistic analyses," *European Journal of Operational Research*, vol. 15, no. 1, pp. 100–109, 1984.

[33] *Intel QuickPath Interconnect*, http://www.intel.com/technology/quickpath/.

[34] *Nallatech: Intel Xeon FSB-FPGA Accelerator Module*, http://www.nallatech.com/Intel-Xeon-FSB-Socket-Fillers/fsb-development-systems.html.

[35] D. J. Miller, P. M. Watts, and A. W. Moore, "Motivating future interconnects: a differential measurement analysis of pci latency," in *Proceedings of the 5th ACM/IEEE Symposium on Architectures for Networking and Communications Systems*, ser. ANCS '09. New York, NY, USA: ACM, 2009, pp. 94–103.

[36] J. Cong and Y. Zou, "Fpga-based hardware acceleration of lithographic aerial image simulation," *ACM Trans. Reconfigurable Technol. Syst.*, vol. 2, pp. 17:1–17:29, September 2009.

[37] G. Estrin, "Organization of computer systems: the fixed plus variable structure computer," in *Papers presented at the May 3-5, 1960, western joint IRE-AIEE-ACM computer conference*, ser. IRE-AIEE-ACM '60 (Western). New York, NY, USA: ACM, 1960, pp. 33–40.

[38] J. Vuillemin, P. Bertin, D. Roncin, M. Shand, H. Touati, and P. Boucard, "Programmable active memories: Reconfigurable systems come of age," *IEEE Transactions on VLSI Systems*, vol. 4, pp. 56–69, 1996.

[39] J. Hauser and J. Wawrzynek, "Garp: a mips processor with a reconfigurable coprocessor," *Field-Programmable Custom Computing Machines, Annual IEEE Symposium on*, vol. 0, p. 12, 1997.

[40] P. Stillwell, V. Chadha, O. Tickoo, S. Zhang, R. Illikkal, R. Iyer, and D. Newell, "Hippai: High performance portable accelerator interface for socs," in *High Performance Computing (HiPC), 2009 International Conference on*, Dec 2009, pp. 109–118.

[41] J. Cong, M. A. Ghodrat, M. Gill, C. Liu, G. Reinman, and Y. Zou, "Axr-cmp: Architecture support in accelerator-rich cmps," in *Proceedings of the 2nd Workshop on SoC Architecture, Accelerators, and Workloads (SAW-2)*, San Antonio, TX, Feb 2011.

[42] C. Lee and Z. Salcic, "A fully-hardware-type maximum-parallel architecture for kalman tracking filter in fpgas," in *Information, Communications and Signal Processing, 1997. ICICS., Proceedings of 1997 International Conference on*, vol. 2, Sep 1997, pp. 1243–1247.

[43] V. Bonato, E. Marques, and G. Constantinides, "A parallel hardware architecture for scale and rotation invariant feature detection," *Circuits and Systems for Video Technology, IEEE Transactions on*, vol. 18, no. 12, pp. 1703–1712, Dec 2008.

[44] S. E. Lee, Y. Zhang, Z. Fang, S. Srinivasan, R. Iyer, and D. Newell, "Accelerating mobile augmented reality on a handheld platform," in *Computer Design, 2009. ICCD 2009. IEEE International Conference on*, Oct 2009, pp. 419–426.