Semi-Dense SLAM on an FPGA SoC

Konstantinos Boikos
Department of Electrical and Electronic Engineering
Imperial College London
Email: k.boikos14@imperial.ac.uk

Christos-Savvas Bouganis
Department of Electrical and Electronic Engineering
Imperial College London
Email: christos-savvas.bouganis@imperial.ac.uk

***Abstract*—Deploying advanced Simultaneous Localisation and Mapping, or SLAM, algorithms in autonomous low-power robotics will enable emerging new applications which require an accurate and information rich reconstruction of the environment. This has not been achieved so far because accuracy and dense 3D reconstruction come with a high computational complexity. This paper discusses custom hardware design on a novel platform for embedded SLAM, an FPGA-SoC, combining an embedded CPU and programmable logic on the same chip. The use of programmable logic, tightly integrated with an efficient multicore embedded CPU stands to provide an effective solution to this problem. In this work an average framerate of more than 4 frames/second for a resolution of 320x240 has been achieved with an estimated power of less than 1 Watt for the custom hardware. In comparison to the software-only version, running on a dual-core ARM processor, an acceleration of 2× has been achieved for LSD-SLAM, without any compromise in the quality of the result.**

I. Introduction

Simultaneous localisation and mapping, or SLAM, has been an important and active research area in the field of robotics for more than two decades. It is fundamental for autonomous robotics that can operate in unknown environments in the real world, such as self-driving cars and UAVs, and can open the way to exciting applications. Existing work includes using swarms for rapid exploration of large spaces [1], much more effective search and rescue operations [2] and precision agriculture.

UAVs and other robots impose tight constraints on the power and weight of the electronics on board. Other applications that require some form of SLAM, such as augmented reality, are also power and performance constrained. Research and industry solutions for SLAM in the embedded space are either using onboard processing and extremely efficient software implementations for low power platforms, or attempt to offload the computationally demanding tasks to a base station. In the first case, the usual approach is to trade information richness and accuracy for performance [3]. However, emerging applications in robotics require unprecedented information richness and accuracy, which these solutions cannot provide.

In the second case, a base station offers larger processing power, and allows more advanced algorithms to be used [4], but this comes with many disadvantages as well. Not only does it limit the range of operation significantly, SLAM applications would require uninterrupted, high bandwidth and low latency communication. Hence, for practical applications on embedded platforms, computation has to be on-board, and with a high computation per watt ratio.

This amount of computational demand, not met by software optimisation, could potentially be met by new reconfigurable hardware platforms. FPGAs lately have had many improvements on the power front, new tools promise improvements in programmability and FPGA SoCs that combine FPGAs with mobile CPUs on the same chip make integration in lightweight embedded platforms much easier as well. This type of heterogeneous system, while having low enough power requirements to fit low-power platforms, stands to substantially improve performance and bring more advanced and computationally demanding algorithms to the embedded space. Additionally, the low and deterministic latency characteristics of FPGA systems offer another unique advantage for many SLAM applications.

This work investigates the performance and acceleration opportunities for a complex, optimised, state-of-the-art Semi-Dense SLAM algorithm, based on LSD-SLAM by Engel et al. [5] using an FPGA SoC. The implementation that was used as a base is the one the authors provide for research use, and will be discussed in more detail in section IV. The main contributions of this work are: the first, to the best of our knowledge, mapping of the LSD-SLAM algorithm on an FPGA, and the identification of opportunities and bottlenecks of heterogeneous FPGA SoCs for such algorithms. The next two sections will give an overview of related work, and a background for the rest of the paper.

II. Background

SLAM is an inverse problem, in which the goal is to recover as much information as possible about an environment using a series of noisy observations, insufficient to produce a unique solution, simultaneously solving two interdependent probabilistic problems, determining the uncertain position of the observer as well as the uncertain position of the observations relative to it. As a result of this, most algorithms are based on complex probabilistic models that in conjunction with the large amount of data required to construct a 3D-map result in high computational requirements. As a result, they require at the very least a modern multicore desktop CPU to run in real time, and often high-end GPUs for acceleration as well [5], [6].

In describing SLAM algorithms, we can break them down to Tracking, Mapping and sometimes Global Optimisation.

Tracking usually refers to the frame to frame, real-time estimation of the camera pose, and feature selection and detection. Mapping refers to the use of that pose and features for landmark depth estimation and subsequently the reconstruction of the environment in two or three dimensions. Finally, Global Optimisation refers to the joint optimisation of a series of poses and observations, to minimise the total accumulated error. The state of the art in SLAM uses a set of high quality frames called keyframes as the basis of mapping and bundle adjustment, a form of semi-global optimisation, which allows to extend the optimisation to a larger trajectory for less computational resources.

SLAM algorithms are defined as Sparse when they are using a comparatively sparse set of observations for tracking and maintain a sparser map of the environment. On the other end, Dense SLAM algorithms attempt to reconstruct most of the environment in their visual field, and often use more of their observations for tracking as well. Lately, algorithms are described as Semi-Dense when they don't necessarily deal with all of the visual information at their disposal, but they use a lot more observations than typical Sparse SLAM algorithms. Semi-Dense SLAM is an attempt at generating more useful maps for robotics by including a larger amount of points in the depth map calculated, while retaining some of the efficiency of only processing a high quality subset of the available visual information.

Consequently, it is not straightforward to compare the performance of different systems, since SLAM is comprised of a set of tasks, the computational complexity of which is strongly dependent on the amount of information they process. That is itself related to many parameters, from frame resolution to point selection threshold and reconstruction density, which directly impact the accuracy and information richness of the resulting localisation and mapping.

III. Related Work

SLAM implementations in the embedded space were traditionally dominated by sparse and mainly feature-based SLAM methods due to power and performance constraints in these platforms, often using heuristics and reducing the problem to two dimensions [3]. Other times algorithms focus entirely on robust tracking and do not maintain a global map, at which point they are classified as Visual Odometry or V.O., as for example in [7]. Building upon V.O. systems, another approach in the literature is using off-board processing on base stations for dense mapping and global optimisation in conjuction with lightweight onboard localisation methods [4], [8]. There are however, significant drawbacks to this approach, which include increased power consumption, high bandwidth requirements and reduced autonomy and area of operation.

Most popular feature detectors used in SLAM systems, have been offloaded to FPGAs in the past, as for example in [9], however there has not been much work in modern, advanced SLAM systems on reconfigurable logic. A recent example of a hardware system combining a mobile CPU and FPGA for real-time SLAM is [10], where an FPGA acts as a preprocessor to perform disparity estimation, and streams its results towards the main memory. In a comparison paper [11], this work is compared with a software based system for real-time vision on a high-speed fixed-wing aircraft, running on a pair of embedded CPUs. They report that the FPGA achieves a significantly more dense depth map, even though it has a quarter of the power consumption.

One of the best performing and well-known SLAM systems is LSD-SLAM [5]. In comparison to other state-of-the-art methods, it is the only one to combine robust, accurate and efficient direct photometric tracking, with a semi-dense map as output, a very important feature for robotics. According to the authors, skipping the computationally heavy step of feature extraction improves efficiency and also means it works well in a variety of environments.

IV. LSD-SLAM

As its tracking method, LSD-SLAM performs direct whole-image alignment to recover the camera pose, and subsequently performs a filtering depth estimation to generate depth information. It incorporates uncertainties about the depth estimation in both tracking and mapping. After a sufficient camera displacement, a new keyframe is generated and the previous keyframe is retired and incorporated in a pose-graph, which for LSD-SLAM represents the global map.

LSD-SLAM uses a subset of the frame pixels, under the assumption that the most useful pixel areas are those characterised by a high intensity gradient, and therefore some kind of texture or edge. The pose is recovered by aligning the frame that is tracked with the current keyframe, to minimise the photometric error. The optimisation method used is Levenberg-Marquardt, applied at the variance-normalized sum of the photometric errors, calculated by directly comparing pixel intensities. This irregular pattern of data accesses in tracking is one of the unique challenges in mapping this type of algorithm well to custom hardware.

Depth is estimated and subsequently refined by filtering over many per-pixel small-baseline stereo comparisons [5], making a high processing frame-rate crucial to support fast movement for the robot. The data structure to store these is the keyframe itself, which includes the pixel values of the original image, and for the subset of the pixels considered reference points, depth and depth variance information as well. The information stored in the keyframe has to be used for every new frame being tracked, and is accessed in a non-serial fashion, making memory and caching an important consideration.

A. Profiling and Timing Results

The source code used as a base is the one provided by the authors of the original paper [5] for research purposes. It consists of highly optimised C++ with SIMD instructions, and was tested, profiled and timed on a desktop and a dual core ARM Cortex-A9. The main timing results can be seen in Table I. Profiling identified the functions that took up a significant amount of time, but there was not one specific bottleneck. Both mapping and tracking consisted of interdependent worker

functions which shared the work, and there was significant communication between the threads as well.

TABLE I
TIMING RESULTS

Task& Execution Time	Desktop CPU	ARM
Tracking	12 ms	440 ms
Global Optimisation	20 ms	128 ms
Mapping	13 ms	576 ms

The results indicate that the ARM processor's performance is more significantly impacted by the large amount of memory transactions in this algorithm. Both tracking and mapping are very computationally demanding. However, the tracking is the front-end of the algorithm, delivering its results to the mapping task to process, and the quality and performance of this task is crucial for the whole algorithm to function robustly. Moreover, fast and accurate tracking is crucial in SLAM, and must be established before accurate mapping can be introduced. This is especially true for LSD-SLAM where tracking converges only for a small frame to frame baseline. Because of these results, it was decided to first offload the tracking pipeline to the FPGA, keeping the rest of the software and the high-level control on the CPU.

V. SYSTEM ARCHITECTURE

The System Architecture is shown in Fig. 1. Two accelerators were implemented to offload the tracking task on the FPGA, a residual and weight calculation unit and a jacobian update unit. They were connected through an AXI-BUS interface to the ARM CPU which handled control, as well as a dedicated high-performance memory port which issues transactions directly to the off-chip RAM. Both the CPU and the custom hardware have access to the same memory space and a lot of redundant copying is avoided. The custom hardware was designed to maintain both accuracy and compatibility with the software implementation and to interface with the rest of the algorithm running as software on the ARM CPU. Hence, most calculations are on single-precision floating-point numbers, and some intermediate results are stored as double-precision. The reason two units were used in the FPGA is to simplify the first unit's design and control logic, as the Jacobian Update unit is called for only a subset of the iterations.

A. Residual and Weight calculation Unit

In this unit there is one block pre-fetching batches of inputs, one block writing back batches of outputs and the main calculation block which performs all of the necessary computation. Before the unit begins work, the entire frame being tracked is pre-fetched in a local frame buffer. This is done because pixel gradients are required in a random pattern, and would require many thousands of unstructured memory read requests. In the original algorithm, they would be calculated before tracking begun and stored in the DRAM. Instead, by pre-fetching pixel intensity information and calculating these quantities on the fly, the hardware pipeline's throughput was increased by more than a factor of two as tested.

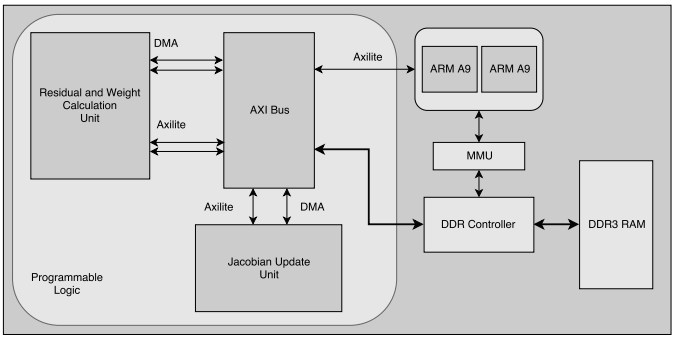

Fig. 1. System Architecture

This happens for two reasons. The size of the data loaded is much smaller, and by performing one large burst read from memory the overall latency cost is much smaller in comparison to many random access requests. This design increases performance and power efficiency at the same time since off-chip memory requests are significantly reduced. The unit's inputs and outputs are also read and written as buffered bursts rather than sequentially for every iteration, again to take advantage of the efficiency of burst reads and writes to reduce the overall cost of the memory access latency.

B. Jacobian Update Unit

This second accelerator calculates the derivative values for every single reference pixel, and adds them to calculate the Jacobian, as well as the Hessian approximation $\boldsymbol{H} \approx \boldsymbol{J}^T \boldsymbol{J}$. A weight factor is also included in the calculation. It reads back the last version of the buffers produced by the other accelerator when the linear system solution is accepted as one that reduced the error, and calculates the Jacobian and Hessian by accumulating the update for each tracked point.

VI. EVALUATION

The SLAM System was implemented and tested on a Zedboard, with a Zynq-7020 SoC. Vivado HLS was used to implement the two accelerators which were then imported in Vivado to interface and synthesise the entire system. Running on the ARM CPU is an Ubuntu-based Linux OS which supports the software, as well as custom drivers to interface with the hardware accelerators.

The FPGA was run at a clock frequency of 100 MHz, to satisfy the timing of the critical path. At that frequency the system achieved on average a total frame processing time, including the cost of memory copy and some bookkeeping on the software side, of 218ms per frame. This corresponds to a framerate of on average 4.55 fps for our main test scene, tracking at a resolution of 320x240. For the exact same test sequence the software only version running on the dual-core ARM A9 achieved 2.27 fps. Additionally, the FPGA accelerated system was consistently $2\times$ faster for a variety of scenes. This performance was achieved with an estimated

dynamic power consumption of 0.71 Watts for the FPGA, and a chip total of approximately 2.25W. The power consumption of the software-only version was estimated at 1.54 Watts for 50% average core load, as a conservative estimate. Crucially, even with these figures the FPGA can perform the entire tracking at an estimated 6.40 frames/Joule in comparison to just 1.48 frames/Joule for the CPU.

For the FPGA that was used, the resources consumed were 54% for the DSPs (119 DSPs) and 44.7% for the Flip-Flops (47.5K), with a design not aggressively optimised for area. In order to interface custom hardware with software that uses large dynamically allocated buffers, in scattered virtual pages, the system spends some time on transferring the input data to a dedicated DRAM region. The copy of input data to a dedicated area of the DDR memory takes a significant percentage of the total execution time, up to 25%, depending on the execution level. However the bottleneck comes from a combination of this, and memory latency and bandwidth, since it is necessary to issue non-consecutive transactions to and from different memory regions, to maintain compatibility and interface with the software.

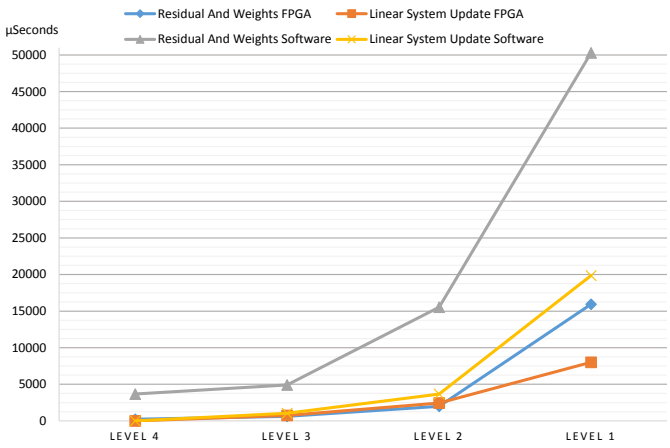

Fig. 2. Software and Hardware timing for coarse-to-fine Pyramid Levels

The per function performance, of the hardware accelerator scales linearly based on the number of points, with a small extra penalty on the final level due to the additional random reads and writes necessary to support the original software implementation. This is shown in figure 2. Residual and Weights refers to the calculation of the photometric residual, and the residual weights, two different functions in software. Linear System Update refers to calling the Jacobian update unit to calculate the Jacobian and Hessian, and then solving the linear system. It is important to note that the residual and weight calculation unit is called on average three to four times as often as the linear system update, while the memory copy mentioned in the previous paragraph happens only once per pyramid level.

VII. Conclusion

We have presented an efficient mapping of LSD-SLAM, a state-of-the-art, Semi-Dense SLAM algorithm on an FPGA System on Chip, achieving a $2\times$ acceleration and more than $4.3\times$ the energy efficiency, measured in frames/Joule, compared to a software version running on an embedded CPU. The system presented is capable of running LSD-SLAM, while tracking at more than 4 fps at a resolution of 320x240, limited only by memory latency and bandwidth, with an estimated power consumption of less than 2.5 Watts for the SoC. This work has demonstrated that an FPGA SoC is capable of bringing advanced and more dense SLAM algorithms to embedded low-power devices, with increased performance and greater performance-to-watt ratio than other solutions.

Future work with FPGA SoCs should begin with hardware/software co-design because of the nature of these algorithms, placing a lot of importance on the memory architecture, including caching techniques and data movement, to ensure scalability, and compatibility with dense or higher-resolution algorithms. A redesign of software implementations to allow a streaming interface with the FPGA, with a large guaranteed bandwidth and the CPU free to perform higher-level optimisation and control is ideal.

References

[1] M. Saska, J. Chudoba, L. Precil, J. Thomas, G. Loianno, A. Tresnak, V. Vonasek, and V. Kumar, "Autonomous deployment of swarms of micro-aerial vehicles in cooperative surveillance," in *Unmanned Aircraft Systems (ICUAS), 2014 International Conference on*. IEEE, 2014, pp. 584–595.

[2] S. Waharte and N. Trigoni, "Supporting search and rescue operations with UAVs," in *Emerging Security Technologies (EST), 2010 International Conference on*. IEEE, 2010, pp. 142–147.

[3] S. Lee, S. Lee, and J. J. Yoon, "Illumination-invariant localization based on upward looking scenes for low-cost indoor robots," *Advanced Robotics*, vol. 26, no. 13, pp. 1443–1469, 2012.

[4] J. Sturm, E. Bylow, C. Kerl, F. Kahl, and D. Cremers, "Dense tracking and mapping with a quadrocopter," in *UAV-g 2013*, 2013.

[5] J. Engel, T. Schöps, and D. Cremers, "LSD-SLAM: Large-scale direct monocular SLAM," in *European Conference on Computer Vision (ECCV)*, September 2014.

[6] R. A. Newcombe, S. J. Lovegrove, and A. J. Davison, "DTAM: Dense tracking and mapping in real-time," in *Computer Vision (ICCV), 2011 IEEE International Conference on*. IEEE, 2011, pp. 2320–2327.

[7] C. Forster, M. Pizzoli, and D. Scaramuzza, "SVO: Fast semi-direct monocular visual odometry," in *Robotics and Automation (ICRA), 2014 IEEE International Conference on*. IEEE, 2014, pp. 15–22.

[8] J. Engel, J. Sturm, and D. Cremers, "Camera-based navigation of a low-cost quadrocopter," in *Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on*. IEEE, 2012, pp. 2815–2821.

[9] D. Bouris, A. Nikitakis, and I. Papaefstathiou, "Fast and efficient FPGA-based feature detection employing the SURF algorithm," in *Field-Programmable Custom Computing Machines (FCCM), 2010 18th IEEE Annual International Symposium on*. IEEE, 2010, pp. 3–10.

[10] D. Honegger, H. Oleynikova, and M. Pollefeys, "Real-time and Low Latency Embedded Computer Vision Hardware Based on a Combination of FPGA and Mobile CPU," in *Intelligent Robots and Systems (IROS 2014), 2014 IEEE/RSJ International Conference on*. IEEE, 2014, pp. 4930–4935.

[11] A. J. Barry, H. Oleynikova, D. Honegger, M. Pollefeys, and R. Tedrake, "Fast Onboard Stereo Vision for UAVs," in *Vision-based Control and Navigation of Small Lightweight UAV Workshop, International Conference On Intelligent Robots and Systems (IROS)*, 2015.