MobileSLAM: SLAM for Low Power Embedded System

ABSTRACT

Simultaneous Localization and Mapping(SLAM) is important for many robotic vision and argument reality tasks. It is essential but challenging to adapt SLAM on mobile platforms because of the gap between the application demands and the hardware capabilities. In this work, we first perform workload characterization for SLAM applications. We discover that there are abundant data parallelism existing in the application and the memory bandwidth requirements vary across program phases. Based on the observation, we suggest possible insights for hardware design, and implement MobileSLAM on a commercial-on-shelf mobile platform Nvidia Tegra TX1. The results show that the MobileSLAM implementation on Tegra TX1 can achieve comparable performance as baseline implementation on a desktop 4-core Intel CPU running at 3.4GH, with a significant power reduction (6x). Compare to ORB-SLAM, the state-of-art SLAM system, MobileSLAM has a performance improvement of 1.5x with the same power consumption on the mobile platform.

CCS CONCEPTS

• **Computer systems organization** → *Heterogeneous (hybrid) systems*; • **Software and its engineering** → Embedded software;

KEYWORDS

SLAM, Embedded platform, Workload characterization, Heterogeneous architecture.

ACM Reference Format:

. 1997. MobileSLAM: SLAM for Low Power Embedded System. In *Proceedings of ACM Woodstock conference (WOODSTOCK'97),* Jennifer B. Sartor, Theo D'Hondt, and Wolfgang De Meuter (Eds.). ACM, New York, NY, USA, Article 4, 6 pages. https://doi.org/10.475/123_4

1 INTRODUCTION

Simultaneous localization and mapping (SLAM) is one of the key functions in robotic vision. SLAM system constructs the map of an unknown environment while keeps tracking of the location. It is an essential function in many robotic application scenarios, such as indoor vacuum machines, augmented reality devices, drones, and autonomous vehicles [6, 7].

Implementing SLAM with advanced sensor systems such as lasers or sonar to build 2D maps of small static environments is considered to be straightforward and relatively easy[6, 7]. However, implementing SLAM for dynamic, complex, and large scale environments, using cameras as the sole external sensor, is an active area of research. Consequently, We focus on the visual-based SLAM approach, using monocular/stereo cameras or with depth sensor, instead of large and costly sensors such as laser range finder and LiDAR, which are not applicable in embedded devices. Visual-based SLAM refers to the SLAM system that uses cameras as the only source of external information, in order to establish the position of a robot or a vehicle in an environment, constructing a representation of the explored zone. The computer vision techniques employed in visual SLAM, such as detection, description and matching of salient features, image recognition and retrieval, among others, are very active research topics in robotic community [1]. However, most of the research are still based on SLAM implementations on powerful computation platform such as desktop CPU or GPU, rather than on low-power embedded computation platform where the computation capability and power consumptions are major constraints.

Enabling low-power embedded devices with visual-based SLAM is challenging. First, SLAM applications have complex pipelines with intensive computations and irregular data accesses, and need to meet the stringent real-time requirements. Second, to support visual-based SLAM on the battery-powered mobile devices with limited power budget and computing resources, there is a huge gap between the computing-demand of the SLAM application and the performance/power capability of the embedded hardware platforms, because the embedded CPU/GPU has much lower power budget and worse performance compared to their desktop counterparts.

In this study, we first perform the workload characterization for two state-of-the-art visual-based SLAM applications, ORB-SLAM [6] and LSD-SLAM [3], to identify the performance bottlenecks. Based on the characterization, we propose a task allocation method to map SLAM functions on a heterogeneous embedded platforms integrating a CPU and a GPU. The main contributions are as follows.

- We characterize ORB-SLAM and LSD-SLAM applications and identify the performance bottlenecks. Based on the observation, we give insights of how to design mobile hardware platform for SLAM applications.
- We present MobileSLAM, a software framework for SLAM to work on commercial embedded platforms, which achieves performance comparable to CPU while consuming much less energy.

2 BACKGROUND

SLAM is the key function in mobile robot localization. The goal of SLAM is to retrieve as much as information as possible about an environment using a series of noisy observations. The architecture of a SLAM system includes two main components: the front-end and back-end as shown in Figure 1[**?**]. The front-end abstracts sensor data into models for estimation, while the back-end performs inference on the abstracted data produced by the front-end. The back-end feeds back information to the front-end to support loop closure detection and validation.

There are two trends in SLAM research work recently. The first one is going from sparse representation to dense representation[**?**]. Sparse representations are far from enough in robotic vision which

WOODSTOCK'97, July 1997, El Paso, Texas USA

ACM ISBN 123-4567-24-567/08/06…$15.00
https://doi.org/10.475/123_4

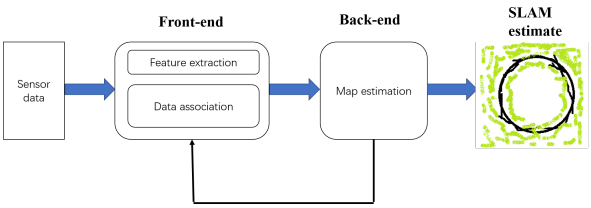

Figure 1: Front-end and Back-end in a typical SLAM system

requires interaction with the environment. The second trend is going from metric-based approach to semantic-based approach[?]. Visual SLAM is currently shifting from mere metric to semantic, thus requires more detailed pose estimation of objects in order to present more semantically meaningful map for SLAM systems. The technical trends require more computational and memory resources which are beyond of mobile platform capability. ORB-SLAM is the state-of-art work of feature-based SLAM systems which has the lightest workload on both computation and memory [4][9]. However, it's still not possible for ORB-SLAM to run on embedded platforms with only ARM CPU. In our analysis, ORB-SLAM only has 6 FPS running on embedded ARM CPU which is far from real-time constraints.

3 WORKLOAD CHARACTERIZATION FOR SLAM

To adapt SLAM on mobile platform, the first task is to characterize the compute pattern and memory access pattern of SLAM systems in order to design hardware acceleration for embedded platforms. We chose ORB-SLAM [6] and LSD-SLAM [3] for characterization. All tests evaluated in this section are on a desktop with a 4-core Intel(R) CORE(TM) i7-4770 CPU running at 3.4GHz frequency, taking the TUM dataset as input unless specially mentioned.

3.1 Compute Pattern

We analyze the compute pattern of SLAM objects in terms of function and instruction characteristics. We have the observation that abundant vector operations exist in SLAM applications which are perfect match for mobile GPUs.

Front-end: First, we trace and analyze the front-end functions in SLAM systems. In feature-based SLAM systems, such as ORB-SLAM, Oriented FAST is used for feature extraction and BRIEF as its descriptor. Oriented FAST is capable of quick corner detection to check whether a pixel has influence difference with the surroundings[8]. BRIEF is a vector descriptor consisting of binary bits. In the matching part, image t has a set of feature points $I_t = \{X_1, X_2, ..., X_n\}$ and image p will have its set $P_t = \{Y_1, Y_2, ..., Y_m\}$. The easiest way of finding matching points is Brute-Force Match in which the hamming distance of each element in I_t to P_t will be calculated to find the smallest as the match one. If each set has a hundred elements, ten thousands of vector comparison will be needed to finish the task between two frames which is an intensive task for execution on CPU. It is important to optimize software for mobile GPU in embedded platforms.

Same conclusion also applies to direct method SLAM like LSD-SLAM[3]. Instead of extracting feature points, LSD-SLAM uses direct method to build semi-dense map which has even larger input data volume than the feature-based methods. It is impossible for most dense SLAM systems to achieve real-time processing on CPU.

Back-end: Second, we analyze the back-end of SLAM systems. Both feature-based and direct SLAM systems have the same back-end computation flow, where Bundle Adjustment (BA) is the key problem.

Bundle adjustment refines a visual reconstruction to produce jointly optimal 3D structure and viewing parameters. Bundle refers to the bundle of light rays leaving each 3D feature and converging on each camera center. It is an estimation problem with large parameters in Triggs' view[11]. The purpose of bundle adjustment is to optimize re-projection error in Eq. (1).

$$g(C, X) = \sum_{i=1}^{n} \sum_{j=1}^{m} w_{ij} ||q_{ij} - P(C_i, X_j)||^2 \quad (1)$$

Since Eq. (1) is a non-linear equation which means in real application, attempts will be tried until the threshold are reached. No matter which iteration method we are using, Newton or LevenbergâĂŞMarquardt algorithm, we will have to solve the linear equation Eq. (2). The dimensions of matrix varies with the points matched. However under most situations, several hundreds of feature points will be matched which makes it impossible to solve the equation by finding the inverse matrix of H with the complexity of $O\{n^3\}$ despite the sparse properties founded by Lourakis[5]. Thus, adding SIMD resources or utilizing GPU on embedded system would be a proper way of solving huge computational requirement of SLAM systems.

$$H\Delta x = g \quad (2)$$

Instruction Analysis: We make statistics on instructions based code profiling to verify the conclusion. Fig.2 shows the statistic information of comparison for vector and scalar operations in the most time consuming functions. Vector operations include vaddsdq, vmovapdx, etc. Scalar operations include move, add, sub, etc. It is shown in Fig.2 that in some functions, vector instructions are executed more than scalar ones. Since some instructions define the data movement which is not related to computation characteristics, we remove such operations and the comparison is shown in Fig.3. We can observe that most functions consist of vector operations which require a lot of SIMD resources and perfectly match for GPU devices.

3.2 Memory Access Pattern

SLAM as well as other data-intensive computer vision applications are hitting the memory wall which is the primary bottleneck in conventional computer systems[2]. Data movements consume significant energy in overall energy consumption. To find out the memory access pattern of SLAM systems, we count the DRAM Bandwidth of ORB-SLAM shown in Fig.4 in which 105 epochs were sampled with each epoch to be 0.5s to trace memory bandwidth. After the first period which ORB vocabulary was loaded, DRAM bandwidth starts to go up till the end of ORB-SLAM. The peak bandwidth would be 11.95GB/sec and the smallest value is 0.573GB/sec. The memory bandwidth varies with the stages of SLAM systems from 3GB/sec to 7GB/sec. The reason cause such waveform is that front-end requires

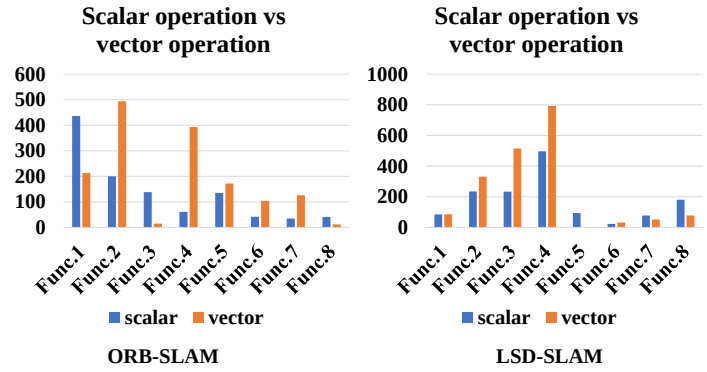

Figure 2: Vector operations and scalar operations comparison for ORB-SLAM and LSD-SLAM

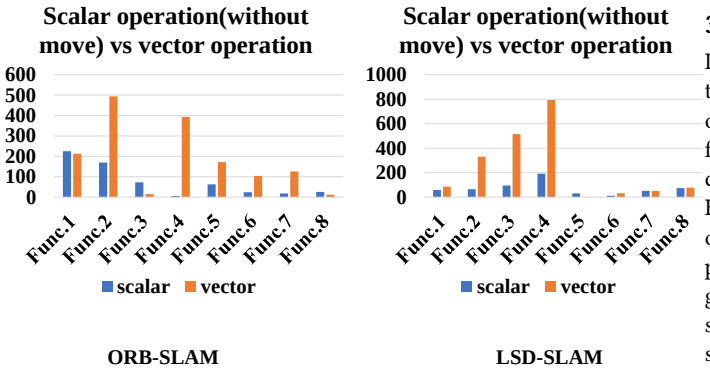

Figure 3: Vector operations and scalar operations comparison for ORB-SLAM and LSD-SLAM without data movement

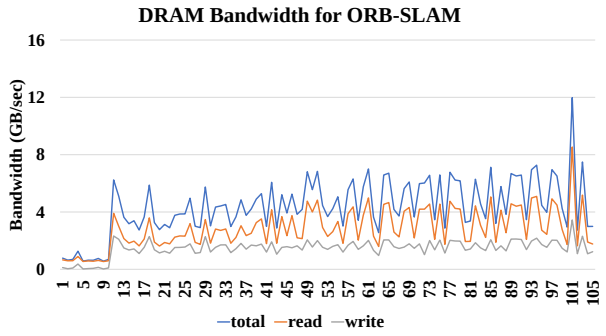

Figure 4: ORB-SLAM DRAM Bandwidth

larger data than back-end. The average is 2.905GB/sec and read requirements are more intensive than write ones. LSD-SLAM has different DRAM access pattern shown in Fig.5. Since LSD-SLAM uses direct method, it has larger average DRAM bandwidth than ORB-SLAM which is 9.7GB/sec. The peak bandwidth goes up to 16GB/sec and smallest one is around 2GB/sec. Also it does not show

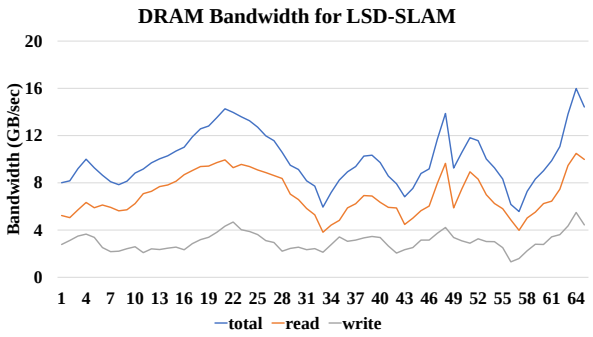

Figure 5: LSD-SLAM DRAM Bandwidth

much rise and fall as ORB-SLAM does since front-end and back-end in LSD-SLAM requires same size of data.

3.3 Insights for Hardware Design

Insights for hardware design of mobile SLAM can be drew based on the characterization we proposed. For both front-end and back-end of SLAM system, huge parallelism exists in compute pattern. For feature-based SLAM works, feature extraction and matching part can be placed on parallel computing devices such as GPUs and FPGAs. Such acceleration methods will also benefits the front-end of direct SLAM systems. Back-end has more complicated compute pattern, most SLAM works utilize third-party library like general graph optimization (g2o) or Ceres. However, such parallelism can still be found in the most computing-intensive part, i.e., the linear solver, which exists in both feature-based and direct method SLAM.

As for addressing the issue of memory bandwidth bottleneck brought from data movement, several approaches can be taken.

- The first one would be data compression. Both feature-based and direct method SLAM use raw figure data took by cameras which add burden to memory access. Data compression can be a way of solving data intensive problem.
- The second would be near-data processing. There is huge difference on memory bandwidth between front-end and back-end for feature-based SLAM works. Front-end requires higher bandwidth since it needs to process the raw input data, while back-end only needs to deal with the data after feature extraction. It will alleviate the memory bandwidth burden when process data immediately after camera caption instead of sending it to back to host.
- The third one would be technologies to increase embedded platform's memory bandwidth, such as Wide I/O[12]. Wide I/O is designed for providing mobile Soc with a maximum amount of bandwidth. Wide I/O 2 will have up to 51.2GB/sec with a 1024-bit interface.

4 OPTIMIZATION

We implement MobileSLAM based on ORB-SLAM systems to utilize the heterogeneous architecture in commercial embedded platforms. ORB-SLAM system is selected because it is one of the state-of-art work on feature-based SLAM. Based on the observations made in Section III, it is important to utilize the embedded GPU to accelerate

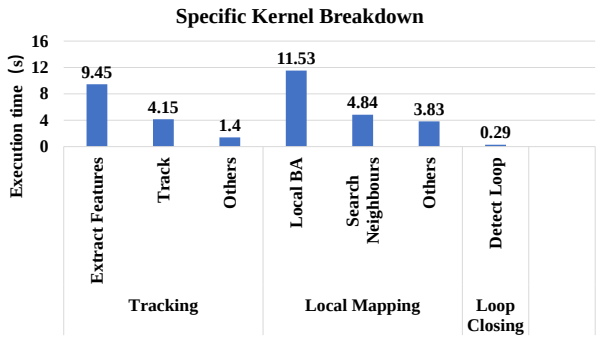

Figure 6: Execution Time Breakdown for ORB-SLAM

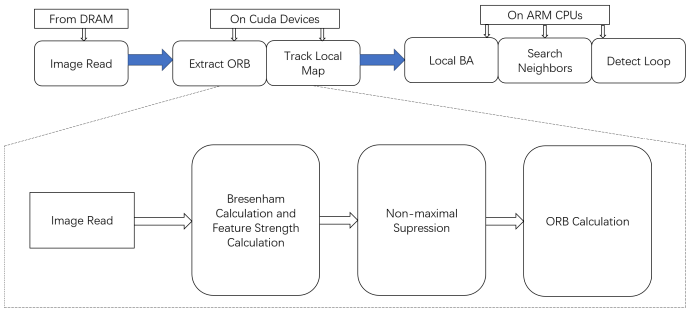

Figure 7: Hardware software matching for ORB-SLAM on TX1

SLAM applications. In this section, we introduce the details of the proposed task offloading method.

4.1 Kernel Breakdown for ORB-SLAM

First, we analyze the breakdown of kernel execution time of ORB-SLAM. The ORB-SLAM mainly consists of three parts: tracking, local mapping, and loop closing. The first two parts occupy nearly 99% of the total running time, while loop closing isn't executed much for under many situations, the routine doesn't contain much loops. Notice that the data association module in SLAM front-end includes a short-term block and a long-term one. The long-term data association (or loop closure) is in charge of associating new measurements to old landmarks. Therefore back-end is decoupled with front-end to support loop closure detection and validation.

Then we track deeper into each kernels to locate the hot spots for the first two stages, as shown in Fig.6. The feature extraction part including FAST corner detection and ORB calculation is the most time consuming part in tracking stage. Local bundle adjustment is the most time consuming part in local mapping stage which takes nearly 20 percent of total execution time.

4.2 Task Mapping Overview

The task mapping is important for performance and resource utilization of heterogeneous architecture. We make function blocks mapping based on several metrics: 1) the vector operation proportion which determines whether it is suitable to map the function block to mobile GPU; 2) the execution time ratio which determines whether this function block is worthy for accelerating; and 3) the data transfer overhead which determines the mapping overhead. We aims to map the most critical and GPU-preferable function blocks to mobile GPU without introducing too much data migration overhead.

Based on the profiling data, one of the most time consuming function blocks is Feature Extraction which consists of FAST corner detection algorithm and ORB feature calculation. As analyzed in previous subsection, the computational pattern indicates that the front-end is suitable for GPU computing. Therefore, the hardware software matching method of our work is shown in Fig.7. Two time consuming part ORB extraction and local map tracking are allocated on CUDA devices to utilize the computation resource from GPU. Other parts are placed on ARM CPU to finish the whole task of tracking and local mapping. In this way, data migration won't be a significant issue since the output of Tracking stage is shrinking enormously.

4.3 ORB Extraction on GPU

Fast and ORB Extraction are the most important parts in front-end, and we will introduce how they are implemented on GPU devices, as shown in Fig.7. After data is read from DRAM, the first block perform Bresenham circle calculation which takes 16 neighboring pixels surrounding the pixel concurrently and measure the difference. Feature strength is calculated by taking the average of sum of all the differences between the pixel. For the second step, non-maximal suppression is performed. If the current pixel being processed has higher feature strength than its neighbors, it become a FAST feature point, otherwise the neighbor with higher strength is the feature points. Afterwards, ORB value is calculated and matching part with previous frame is performed also on CUDA cores. Then the result will be delivered to ARM CPU for performing back-end mapping procedure.

5 EVALUATION

In this section, we analyze the performance and power of MobileSLAM, and compare with the scenarios that original ORB-SLAM executes on desktop and embedded platforms.

5.1 Experimental Setup

We use a commercial-on-shelf platform, Nvidia Tegra TX1, to evaluate performance and power of MobileSLAM and ORB-SLAM on mobile platform. Fig. reftx1 shows the block diagram of TX1, which integrates a quad-core ARM Cortex A-57 CPU and an NVIDIA Maxwell GPU with 256 CUDA cores. Embedded GPU resources can be accessed using different approaches like CUDA and OpenCL. The DMA channels on TX1 can achieve up to 2.5GB/sec bandwidth in total which may be helpful on the intensive data movement in ORB-SLAM. We compare the performance and power under the following three scenarios: ORB-SLAM on desktop with Xeon i7-4770, ORB-SLAM on TX1, and MobileSLAM on TX1.

5.2 Runtime Analysis

We evaluate the execution time of 6 datasets, 3 sequences from TUM datasets and 3 from KITTI datasets, as shown in Fig.9. Under three

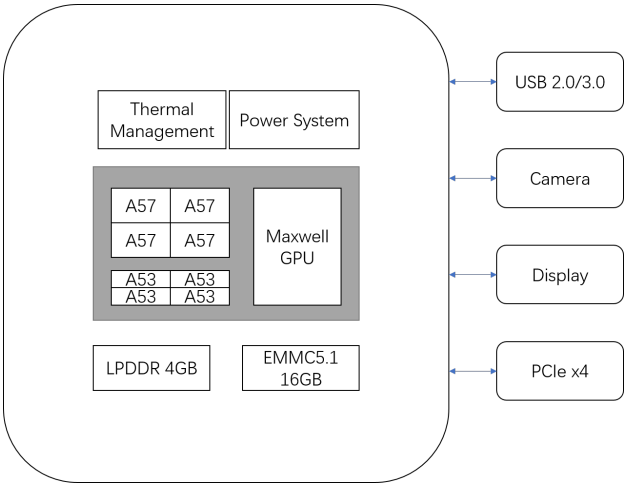

Figure 8: Tegra X1 block diagram

scenarios, the scenario of ORB-SLAM on desktop CPU has the best performance because of the powerful computational capability. The scenario of ORB-SLAM on TX1 has the worst performance which is about 40% slower. The performance of mobileSLAM running on TX1 achieves 0.9x of ORB-SLAM on desktop system.

To evaluate the effectiveness of software-hardware matching strategy of MobileSLAM, we perform kernel analysis on TUM sequence 1. Fig.10 shows the execution time breakdown of kernels including extraction and matching (front-end), local mapping (back-end), and loop closing. As expected, the execution time of ORB extracting and matching is reduced from 9.45s to 4s when MobileSLAM is deployed in the mobile platform. It is observed that back-end performance is also improved, when MobileSLAM allocates the front-end on GPU. As shown in Fig.10, the back-end execution time is reduced by nearly 50%. This is because of the following reasons: 1) the front-end and back-end are coupled for loop closing computation, hence accelerating front-end is beneficial for back-end computation; 2) meanwhile, by offloading front-end computations to GPU, more CPU resources are available to execute back-end steps. In summary, MobileSLAM performs good compared to ORB-SLAM on TX1.

5.3 Power Analysis

To place SLAM systems on embedded robotic platforms, power becomes one of the most crucial design goal. We also evaluate the power consumption on each SLAM datasets, as shown in Fig.11. For indoor datasets TUM1, TUM2 and TUM3, the mobileSLAM achieves 5.2x of power reduction compared to the ORB-SLAM on desktop. The mobileSLAM consumes 20% of more power than ORB-SLAM on TX1 because of the additional GPU computation power. For outdoor datasets KITTI1, KITTI3 and KITTI12, the power consumption compared to CPU would be 6.3x and the same(2.76w to 2.78w) compared to the scenario of ORB-SLAM on TX1. Significant power reduction proved it's possible to place feature based SLAM systems like ORB-SLAM on embedded platforms while utilizing GPU to processing compute intensive tasks.

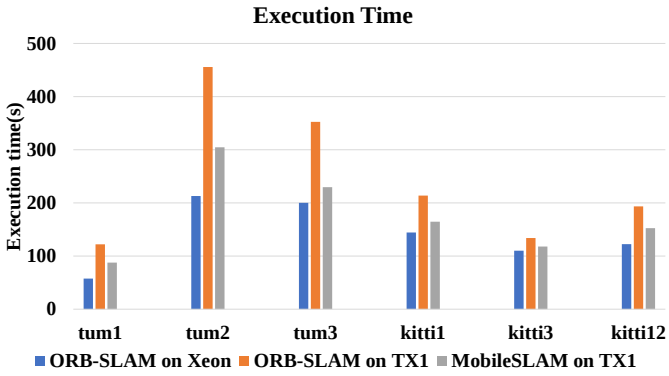

Figure 9: Run time for 6 datasets on 3 version of ORB-SLAM

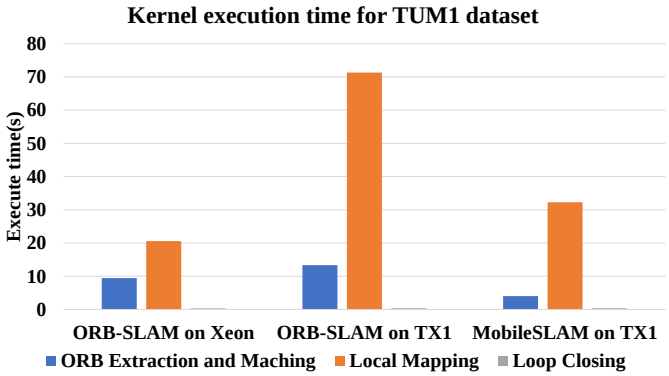

Figure 10: Kernel run time analysis

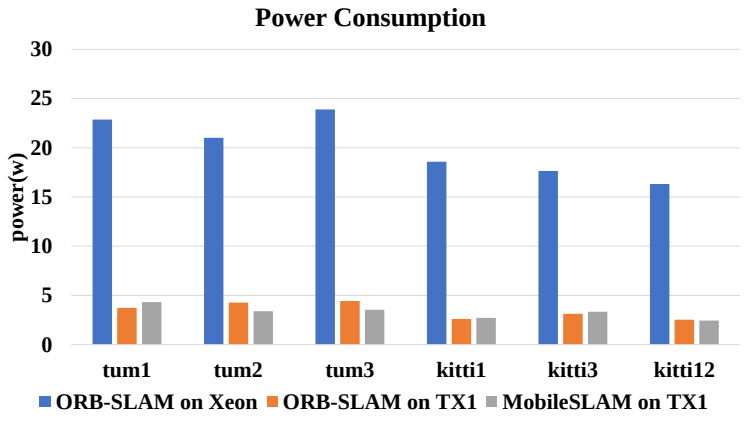

Figure 11: Power Analysis

5.4 Trajectory

Trajectory accuracy is an important functional evaluation factor for SLAM systems. In our work, we used TUM1 which contains 800 images for trajectory accuracy evaluation. The trajectory accuracy is measured by the difference between location distance as in prior work [10]. MobileSLAM has an 0.116m difference in trajectory with

ORB-SLAM on Xeon. However, it is also observed that there are distance difference between ORB-SLAM on Xeon and on TX1. Such inaccuracy is caused by the different precision in different platforms. Compared to the scenario of ORB-SLAM running on TX1, there is only 0.068m difference on trajectory which is negligible.

A HEADINGS IN APPENDICES

The rules about hierarchical headings discussed above for the body of the article are different in the appendices. In the **appendix** environment, the command **section** is used to indicate the start of each Appendix, with alphabetic order designation (i.e., the first is A, the second B, etc.) and a title (if you include one). So, if you need hierarchical structure *within* an Appendix, start with **subsection** as the highest level. Here is an outline of the body of this document in Appendix-appropriate form:

A.1 Introduction

A.2 The Body of the Paper

A.2.1 Type Changes and Special Characters.

A.2.2 Math Equations.

Inline (In-text) Equations.

Display Equations.

A.2.3 Citations.

A.2.4 Tables.

A.2.5 Figures.

A.2.6 Theorem-like Constructs.

A Caveat for the TEX Expert.

A.3 Conclusions

A.4 References

Generated by bibtex from your `.bib` file. Run latex, then bibtex, then latex twice (to resolve references) to create the `.bbl` file. Insert that `.bbl` file into the `.tex` source file and comment out the command `\thebibliography`.

B MORE HELP FOR THE HARDY

Of course, reading the source code is always useful. The file `acmart.pdf` contains both the user guide and the commented code.

ACKNOWLEDGMENTS

The authors would like to thank Dr. Yuhua Li for providing the MATLAB code of the *BEPS* method.

The authors would also like to thank the anonymous referees for their valuable comments and helpful suggestions. The work is supported by the National Natural Science Foundation of China under Grant No.: 61273304 and Young Scientists' Support Program (http://www.nnsf.cn/youngscientists).

REFERENCES

[1] [n. d.]. ([n. d.]).

[2] Brendan Barry, Cormac Brick, Fergal Connor, David Donohoe, David Moloney, Richard Richmond, Martin O'Riordan, and Vasile Toma. 2015. Always-on vision processing unit for mobile applications. *IEEE Micro* 35, 2 (2015), 56–66.

[3] Jakob Engel, Thomas Schöps, and Daniel Cremers. 2014. LSD-SLAM: Large-scale direct monocular SLAM. In *European Conference on Computer Vision*. Springer, 834–849.

[4] Georg Klein and David Murray. 2007. Parallel tracking and mapping for small AR workspaces. In *Mixed and Augmented Reality, 2007. ISMAR 2007. 6th IEEE and ACM International Symposium on*. IEEE, 225–234.

[5] Manolis IA Lourakis and Antonis A Argyros. 2009. SBA: A software package for generic sparse bundle adjustment. *ACM Transactions on Mathematical Software (TOMS)* 36, 1 (2009), 2.

[6] Raul Mur-Artal, Jose Maria Martinez Montiel, and Juan D Tardos. 2015. ORB-SLAM: a versatile and accurate monocular SLAM system. *IEEE Transactions on Robotics* 31, 5 (2015), 1147–1163.

[7] Michał R. Nowicki, Jan Wietrzykowski, and Piotr Skrzypczyński. 2017. Real-Time Visual Place Recognition for Personal Localization on a Mobile Device. *Wireless Personal Communications* 97, 1 (01 Nov 2017), 213–244. https://doi.org/10.1007/s11277-017-4502-y

[8] Edward Rosten and Tom Drummond. 2006. Machine learning for high-speed corner detection. *Computer Vision–ECCV 2006* (2006), 430–443.

[9] Ethan Rublee, Vincent Rabaud, Kurt Konolige, and Gary Bradski. 2011. ORB: An efficient alternative to SIFT or SURF. In *Computer Vision (ICCV), 2011 IEEE international conference on*. IEEE, 2564–2571.

[10] J. Sturm, N. Engelhard, F. Endres, W. Burgard, and D. Cremers. 2012. A benchmark for the evaluation of RGB-D SLAM systems. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*. 573–580. https://doi.org/10.1109/IROS.2012.6385773

[11] Bill Triggs, Philip F McLauchlan, Richard I Hartley, and Andrew W Fitzgibbon. 1999. Bundle adjustmentâĂŤa modern synthesis. In *International workshop on vision algorithms*. Springer, 298–372.

[12] Naoya Watanabe, Masahiro Aoyagi, Tsubasa Bandoh, Takahiko Mitsui, and Eiichi Yamamoto. 2016. Improvement of a TSV Reveal Process Comprising Direct Si/Cu Grinding and Residual Metal Removal. In *Electronic Components and Technology Conference (ECTC), 2016 IEEE 66th*. IEEE, 1259–1264.