Processing-in-SRAM Acceleration for Ultra-Low Power Visual 3D Perception

Yuquan He, Songyun Qu, Gangliang Lin, Cheng Liu, Lei Zhang, Ying Wang*
Institute of Computing Technology, Chinese Academy of Sciences
University of Chinese Academy of Sciences, Beijing, China
{heyuquan20b,qusongyun18z,linganglianг21s,liucheng,zlei,wangying2009}@ict.ac.cn

Abstract

Real-time ego-motion tracking and 3D structural estimation are the fundamental tasks for the ubiquitous cyper-physical systems, and they can be conducted via the state-of-the-art Edge-Based Visual Odometry (EBVO) algorithm. However, the intrinsic data-intensive process of EBVO emplaces a memory-wall hurdle in practical deployment on conventional von-Neumann-style computing systems. In this work, we attempt to leverage SRAM based processing-in-memory (PIM) technique to alleviate such memory-wall bottleneck, so as to optimize the EBVO systematically from the perspectives of the algorithm layer and physical layer. In the algorithm layer, we first investigate the data reuse patterns of the essential computing kernels required for the feature detection and pose estimation steps in EBVO, and propose PIM friendly data layout and computing scheme for each kernel accordingly. We distill the basic logical and arithmetical operations required in the algorithm layer, and in the physical layer, we propose a novel bit-parallel and reconfigurable SRAM-PIM architecture to realize the operations with high computing precision and throughput. Our experimental result shows that the proposed multi-layer optimization allows for high tracking accuracy of EBVO, and it can improve 11x processing speed and reduce 20x energy consumption compared to the CPU implementation.

1 Introduction

Real-time visual simultaneous localization and mapping (vSLAM) is a fundamental technique to unfold the applications of cyber physical systems, such as robotics, autonomous driving, VR/AR, etc. The visual odometry (VO) frontend of a vSLAM system is an indispensable component for the smart devices to estimate the position and orientation of a camera and the 3D structure of the physical environment. Among the various VO models such as feature-based [18], direct [5] or hybrid [7] methods, recently a class of edge-based VO (EBVO) [11, 20, 27] is attracting a greater interest, thanks to its semi-dense reconstruction capability, illumination robustness and large convergence basin. However, EBVO is intrinsically data-intensive which inevitably suffers from the memory-wall bottleneck when deployed onto conventional computing systems. To prove this phenomenon, we leverage Valgrind [19] to profile the computing bottleneck of REVO [20] which is a state-of-the-art EBVO implementation. On a x86 desktop PC, 43% of the total instructions are about data movement, whereas on ARM processors this percentage increases to 51%. Given the fact that data-transfer operations incur 10x more latency and 100x higher energy than arithmetic ones [13], this memory-wall significantly limits the computation throughput and increases energy consumption.

To alleviate such penalties, a promising approach is to embed the computation engine within the memory array, known as Processing-In-Memory (PIM). Compared to other memory devices such as ReRAM [22], DRAM [21] and STT-RAM[10], SRAM based PIM is of particular interest thanks to its mature CMOS manufacture process, low access latency, and almost infinite write endurance. Prior works of SRAM PIMs on domain-specific acceleration, such as CAM [14] or CNN inference engines [3, 23], reshape the memory architecture to perform relatively simple computational kernels. However, for the versatile computing steps of EBVO as shown in Fig.1, designing specific PIM accelerators for each computing kernel is too costly. In contrast, SRAM PIMs also have great potential of general purpose and reconfigurable computing via the bit-serial [4] or bit-parallel [16] solutions. However, these PIM architectures mainly focus on basic logical and arithmetical operations that cannot cover all the computing kernels required by EBVO, and not to mention the lack of study on algorithm-specific data layout and computing pipeline optimization for EBVO.

With this backdrop, we opt to enrich the functionalities of SRAM-PIM for the sake of high-precision EBVO acceleration. To begin with, we analyze the critical steps of EBVO as shown in Fig.1. (a) Edge detection for each input frame. This step incurs low-pass (LPF), high-pass (HPF) image filters and non-maximum suppression (NMS). (b) Estimation of per-frame camera pose via edge alignment. This incurs the warping of every single feature point, and an iterative Levenberg-Marquardt (LM) solver [15] to minimize the warping residual. To obtain high-quality EBVO tracking and 3D reconstruction, the calculation of both steps should be conducted as precise as possible. Based on the above analysis, we propose a novel SRAM-PIM architecture with the systematic support for the EBVO software stack, namely the *algorithm* layer and the *physical* layer.

(1) In the algorithm layer, we propose to leverage PIM friendly computation kernels to realize the two critical steps in EBVO, as well as the optimized data layout and computing schedules for PIM implementation. Specifically, for the edge detection step, we study the featured data reuse patterns of LPF, HPF and NMS, and redesign in-memory pipelines to reduce computation redundancy and data movement in the original algorithm implementation. For the PIM mapping of the per-frame pose estimation step, we adopt high-precision quantization for feature warping and the computation of Jacobian and Hessian

DAC '22, July 10–14, 2022, San Francisco, CA, USA

ACM ISBN 978-1-4503-9142-9/22/07.
https://doi.org/10.1145/3489517.3530446

* Corresponding author.

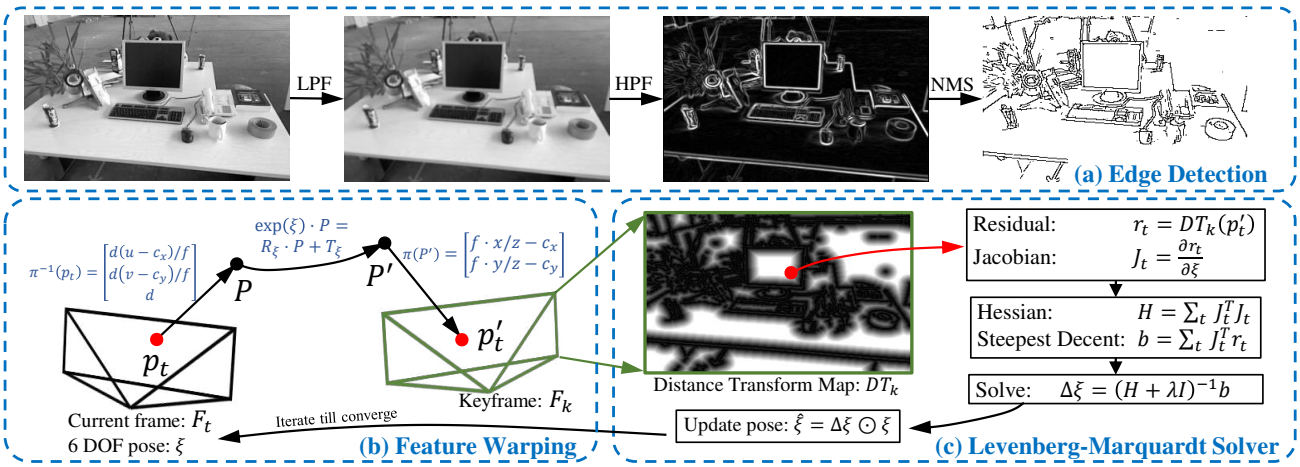

Figure 1: Processing pipeline of EBVO. (a) Edge detection; (b) Feature warping; The detailed formulas are shown in Fig.5-a and Fig.5-b; (c) LM solver; The Jacobian J_t is shown in Fig.5-c.

matrices. We also reschedule the computing phases of the arithmetic operators in EBVO for the sake of data alignment in SRAM and high throughput PIM.

(2) In the physical layer, we propose a bit-parallel SRAM-PIM architecture to realize the algorithmic requirements. We are inspired by the bit-parallel PIM engine with reconfigurable computing precision in [16], and we further enrich the PIM functionalities to perform various operations, such as saturation, min/max and division. On top of this we build up the complicated arithmetic operators of the various computing kernels required by the edge detection and pose estimation in EBVO.

To summarize, this work makes the following contributions:

- We propose a novel processing-in-SRAM accelerator which allows for real-time and ultra-low power EBVO. To our best knowledge, this work is the first demonstration of PIM acceleration in the field of VO/vSLAM.
- To boost the performance systematically, we investigate the PIM friendly algorithm implementation, data mapping and computing schedule for the data-intensive computing kernels of the EBVO stack. On top of this, we distill the essential arithmetical operations that can be efficiently realized in PIM, and we propose an enhanced bit-parallel SRAM-PIM architecture as the hardware backup.
- We perform software simulation using a customized EBVO system and PIM emulator, and hardware simulation using the 90nm CMOS technology. Compared to the baseline embedded MCU of the same CMOS technology, the PIM accelerator achieves an overall 11x acceleration of computing speed with a 20x reduction of power consumption.

2 BACKGROUND

2.1 vSLAM Oriented Accelerators

Hardware accelerators are designed to alleviate the computation burden of specific algorithmic components of VO/vSLAM systems, such as feature detection and state estimation, on resource-constrained platforms. eSLAM [17] accelerates the procedures of ORB feature detection and matching of the ORB-SLAM [18] algorithm on the ZYNQ7000 FPGA platforms, and achieve a much faster tracking speed than the embedded CPU and even a desktop PC. Navion [25] exhibits an ASIC design that accelerates sparse feature tracking and sensor fusion with the inertial measurement unit (IMU), and allows for ultra-low power visual-inertial odometry (VIO) on nano drones. For off-the-shelf products, Intel releases its RealSense T265 tracking camera [9] which allows for near-sensor VIO on the Movidius Myraid 2 VPU. The Microsoft Hololens AR glasses also integrate the HPU [26] accelerator for real-time pose estimation and object rendering. Despite the various FPGA/ASIC based application cases, PIM based non-von-Neumann-style accelerators for VO/vSLAM have not been explored thus far.

2.2 General Purpose SRAM PIM Architectures

The general purpose SRAM PIMs typically leverage the basic architectural prototype [1] that can compute AND/NOR logic directly on the bitlines of SRAM array using two sense amplifiers (SA). To perform versatile arithmetic, the work in [4] proposes bit-serial algorithms for fixed-/float-point add/sub/ mul/div. [8] further explores the more complicated functions such as square root, sin/cos etc. However, the bit-serial designs have high computational complexity, and they also require bit-transpose of operands before and after the computation, which increases the latency as well. By contrast, the work in [16] proposes a bit-parallel PIM that allows for high-throughput and low-power computation of addition and multiplication. [2] compares the performance of bit-serial/parallel schemes, and reveals that both have similar costs of power and area while bit-parallel computation has much lower latency.

2.3 Preliminaries of EBVO

The major task of EBVO is to track the camera pose via edge alignment, as is depicted in Fig.1. To begin with, we denote the current input frame as F_t and the reference keyframe as F_k. The sets of edge features p_t and p_k are detected in F_t and F_k accordingly. During the pose estimation, each feature p_t is warped to F_k, denoted as p'_t, and matched with the nearest p_k to assess the residual. By computing the Jacobian and Hessian matrices, the pose adjustment $\Delta\xi$ is obtained via LM solver to correct the pose of F_t. This process is conducted iteratively to minimize the warp residual, and finally we obtain a

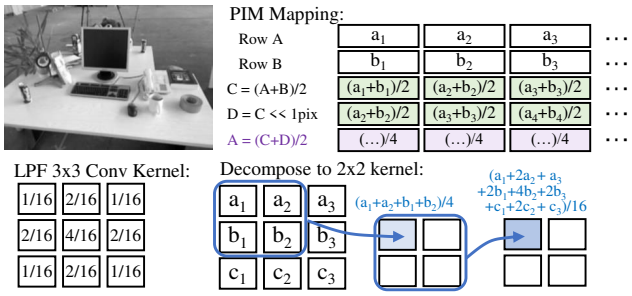

Figure 2: The LPF kernel and its PIM mapping.

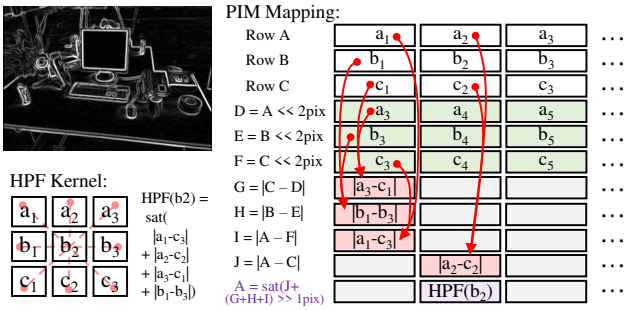

Figure 3: The HPF kernel and its PIM mapping.

high-precision estimate of the position and orientation. To eliminate the overhead of feature matching, EBVO pre-calculate the nearest edge distance throughout the whole image of F_k, known as Distance Transform [6] (DT_k), so that the residual of p'_k can be directly looked-up in DT_k. The gradient of DT is also pre-calculated for F_k, so that part of the Jacobian matrix (see Fig.5) can also be looked-up in this gradient map. At QVGA (320x240) resolution, EBVO typically tracks 3000~6000 features within 10 iterations depending the texture layout of the environment, and this ensures high data parallelism in the feature detection and tracking steps to be exploited for PIM acceleration.

3 Algorithm Layer

In this section we first investigate and simplify the various calculation kernels required for the computing pipeline of EBVO as shown in Fig.1, including the LPF, HPF and NMS kernels for edge detection, and the feature warping and Jacobian kernels for pose estimation.

3.1 Restrictions

The PIM accelerator can be modeled as a SIMD processor with high processing bandwidth aligned to the bit-width of the memory array. However, compared to the powerful SIMD engines of CPU/GPU, PIM can integrate much less functionalities, because in practical chip layout, the shape of the memory cell restricts the PIM computing logic on each word line. The basic operations that PIM could handle in a single clock cycle are typically variants of shift and accumulate, and the complicated operations such as shuffling or branching should be avoided for a PIM friendly computing scheme.

3.2 Edge Detection Kernels

LPF Kernel: LPF is a typical 3x3 convolution throughout the whole image. To perform LPF efficiently, we carefully select a 3x3 kernel whose coefficients are all 2^n, as shown in Fig.2. To exploit the data-reuse patterns, we further decompose it into a pipeline of two simpler 2x2 convolutions, whose coefficients are all 1/4. The PIM

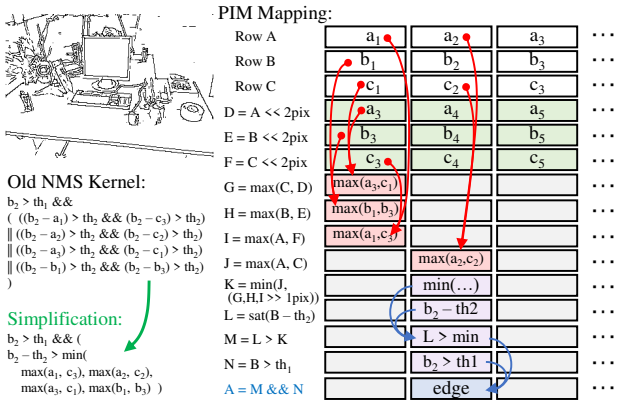

Figure 4: The simplified NMS kernel and its PIM mapping.

mapping of this 2x2 kernel is very efficient, which is also shown in Fig.2. From this LPF kernel, we distill the basic operations: *shifting* and *average*.

HPF Kernel: Traditionally, HPF requires two orthogonal 3x3 Sobel convolution for the gradients g_x and g_y, and then calculates $\sqrt{g_x^2 + g_y^2}$. Obviously this is costly, so we propose an alternative kernel which only calculates the saturated (sat) sum-absolute-difference (SAD) on 4 directions, as shown in Fig.3. This calculation exhibits a similar result to the original Soble kernels, but at a much lower cost. To reduce the overhead of data shuffling, we observe that the SAD operands can be fully aligned by shifting 2 pixels for each row. We also observe the data-reuse pattern that the shifting of rows *B, C* can be pipelined to the next processing. Thereby sat(SAD) can be smoothly calculated, and the HPF kernel requires two new calculations: *saturation* and *absolute difference*.

NMS Kernel: As shown in Fig.4, the original NMS kernel incurs 9 comparison with thresholds th_1 and th_2, as well as a compound of 8 branches. Their naive PIM mappings are obviously inefficient, and to simplify these, we notice the equations: (x>y AND x>z) ⇔ x>max(y,z), and (x>y OR x>z) ⇔ x>min(y,z). Thereby, the original NMS kernel is simplified to only a few *min/max* arithmetical operations, and we will provide a branch-free implementation for them in the next section. Fig.4 also depicts the data-reuse pattern for efficient PIM mapping of the new NMS kernel.

3.3 Feature Quantization and the Warp Kernel

3D Features: As shown in Fig.5-a, a 3D feature is expressed by its pixel coordinate (u, v) and the depth d on the anchor frame. During the iterative pose estimation, the costly operation is to repeatedly warp the features from the current frame to the keyframe. We optimize this by using the quantized inverse-depth coordinate (a, b, c) for the 3D features, which embeds the camera intrinsic parameters f, c_x, c_y. They are quantized to Q4.12 (4 integer and 12 fractional bits) to retain high precision, because our experiment shows that an 8-bit quantization lead to completely fault results, and the 16-bit quantization exhibits a warp error of less than one pixel compared to the float-point calculation.

Warp Kernel: As depicted in Fig.1-b and Fig.5-b, the warped pixel coordinate (u', v') is obtained from a 3D pose translation and camera projection. To quantize this computation, we refer to the elements of 3x3 rotation matrix $r_{ij} \in R$ and 3x1 translation vector $t_i \in T$. They

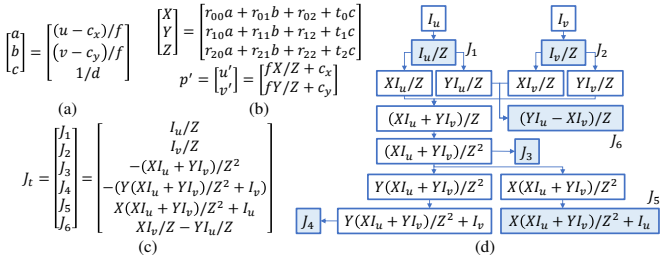

Figure 5: (a) The inverse-depth coordinate for 3D features. (b) The detailed warping function as in Fig.1-b. (c) The Jacobian matrix induced by each feature. (4) The optimized calculation pipeline of the Jacobian kernel.

all range within $(-1, 1)$, because the pose difference is typically small between the adjacent frames. Thereby we have them quantized to Q1.15, which can be aligned to the 16-bit quantized features and be computed efficiently in the PIM array.

3.4 Jacobian and Hessian Kernel

Jacobian Kernel: The Jacobian matrix induced by each warped feature is shown in Fig.5-c, where (I_u, I_v) is the pixel gradient at the warped coordinate (u', v') on the DT_k map, and it is pre-calculated for each keyframe. To reduce computing redundancy, we rearrange the calculation order based on the duplicated items in J_t, and propose a computation pipeline for it as depicted in Fig.5-d. It is quantized to Q14.2.

Hessian Kernel: When each J_t is obtained, we calculate the Hessian part $J_t^T J_t$ and steepest decent part $J_t r_t$, which are in turn added to H and b as in Fig.1 via a typical reduction operation. Finally the LM iteration is finished by solving the pose update $\Delta\xi$ from the 6 DOF linear equations. In our experiment we observe that the LM solver works fine with 32-bit quantized H, whereas a 16-bit quantization would lead to solver failure. Thereby we opt 32-bit Q29.3 for both H and b for the sake of data alignment and high computation precision.

The Jacobian, Hessian and steepest decent matrices can be computed on PIM, because it can process the full set of features in parallel. However, the next step is the linear solver of a small matrix of 6x6, which can hardly benefit from the parallel computing of PIM. Thereby, it should be conducted directly on a CPU without bothering to use PIM acceleration.

4 Physical Layer

4.1 Hardware Architecture

From the algorithm layer, we distill the various basic operations that PIM should support in the physical layer: *logic*, *shifting*, *average*, *saturation*, *absolute difference*, *min/max*, and *mul/div*. To realize them with high precision and efficiency, we adopt the bit-parallel prototype as in [16], and enhance its functionalities with low hardware profile. The proposed architecture is shown in Fig.6.

Memory Array: As a typical configuration for the EBVO, the SRAM array is customized to $(320 * 8) \times 256$ bits, which has enough capacity for an 8-bit QVGA image, or 20480 coefficients of 32-bit for various calculations. This also enables a SIMD throughput of 320x8-bit, 160x16-bit, or 80x32-bit per operation.

Basic PIM Logic: We leverage the basic prototype of in-memory AND/NOR logic (Fig.6-a), along with a NOR gate for logic XOR, and a NOT gate for logic OR. Note that the bit-wise logic XOR and AND

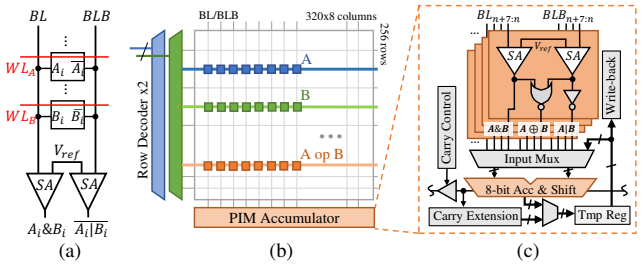

Figure 6: (a) In-memory AND/NOR logic via two sense amplifiers (SA); (b) Bit-parallel SRAM-PIM architecture. (c) A slice of the PIM accumulator.

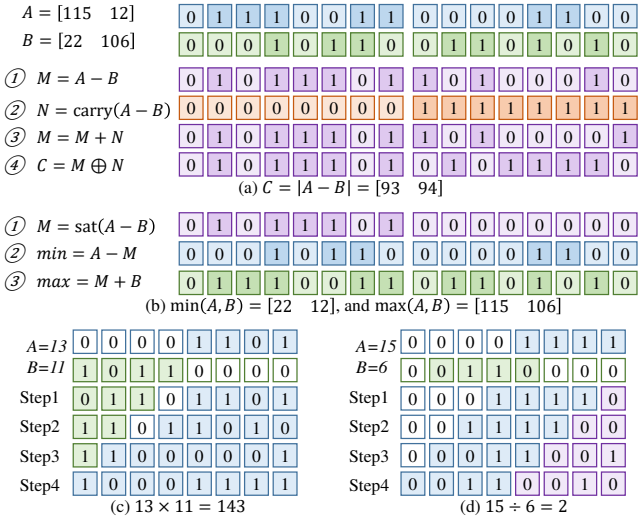

Figure 7: Arithmetic examples: (a) Absolute difference; (b) Min and max; (c) Multiplication; (d) Division and remainder.

perform a half-adder, and they can be accumulated to produce the multi-bit summation.

Digital Arithmetic: As shown in Fig.6-c, the central computing module is the accumulator and shifter grouped in 8-bit slices. Its inputs are multiplexed with the PIM basic logic and the *Tmp Reg*, so as to realize versatile operations. By controlling the propagation of carry bits (*Carry Control*), the computing precision of 8/16/32/64 bit-width can be configured in run-time. The *Carry Extension* generates bitmasks that indicates the summation overflow, which is in turn utilized to perform comparison and saturation.

Result and Write-back: Since the read/write access to the memory cannot coincide with each other, in the computing cycle, the result is stored in *Tmp Reg*. In the next cycle, it can immediately feed the computing logic, or be written back to SRAM. Thereby, the register can stack the temporary result for various multi-stage operations (see Fig.7), so as to alleviate the write-back latency and significantly improve the computational efficiency.

4.2 Arithmetic

Based on this hardware architecture, we can easily realize the trivial operations, such as logic AND/OR, addition $(A+B)$, subtraction $(A+(-B))$, average $((A+B)/2)$ and saturation $(\text{sat}(A+B)$ and $\text{sat}(A-B))$. They are omitted here to simplify the exposition, and we only provide details to the following multi-stage operations.

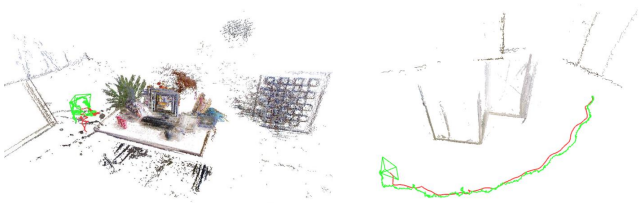

Figure 8: The tracking and reconstruction result of two dataset sequences. The output trajectory (green) can be compared with the groundtruth (red).

Absolute Subtraction: As shown in Fig.7-a, first we obtain $M = A - B$ as well as the carry extension N. Then we can inverse the negative part of M by: $(M + N) \oplus N$.

Min/Max: Inspired by the branch-free algorithm in [12], we obtain $\max(A, B) = \text{sat}(A-B)+B$, and $\min(A, B) = A-\text{sat}(A-B)$, as depicted in Fig.7-b.

Multiplication: Inspired by [16], we obtain multi-bit product $A \times B$ from the MSB to LSB of B as shown in Fig.7-c. Unlike [16] that requires the inefficient bit reverse and multiple shifting, we propose to concatenate both the partial product and B in *Tmp Reg*, so that the bit shifting and summation can be performed simultaneously. This incurs *n+2* clock cycles including the SRAM read/write overhead. Note that both operands should be unsigned, whereas the negative values can be easily inverted before and after the computation.

Division: As shown in Fig.7-d, multi-bit $A \div B$ is conducted via a canonical restoring-division. The partial remainder is stored in *Tmp Reg*, and the quotient bits are stacked in the LSB. Note that both operands should also be unsigned. In this case we obtain arbitrary bit precision of division, and it also incurs *n+2* clock cycles including the SRAM read/write overhead.

5 Evaluation

5.1 Setup

Baseline: We adopt the state-of-the-art PicoVO [11], which is a highly optimized EBVO that can run in real-time on the 216MHz STM32F7 microcontroller. This MCU is manufactured in 90nm technology, which is taken as the baseline CMOS node of our hardware simulation. And we also take the precision, timing and power results of PicoVO on MCU for comparison of accuracy, processing speed and power consumption to our PIM EBVO.

Algorithm and Software: We develop a cycle-accurate PIM simulator in C++ based on the physical design in Section IV to profile the computing cycle of each kernels. We assume that all basic operations are single-cycle, and an extra write-back cycle is required when the output resides in SRAM. We also implement a functional EBVO system that can digest the TUM RGBD dataset [24], and integrate the PIM simulator into it. The drift of the EBVO system is assessed by the root mean square error (RMSE) of relative pose error (RPE) against the groundtruth.

SRAM Array: To estimate the area of 2560x256-bit SRAM array, we take the SRAM model in [4] which has a similar SA layout to our proposal. The SPICE simulation in [4] is conducted in 28nm node, and we scale it to 90nm according to our PIM setup. This reveals an area of 3.48E6 um^2 for memory array, and 5.60E4 um^2 for the SA. The read/write access to the SRAM takes 944.8pJ per operation.

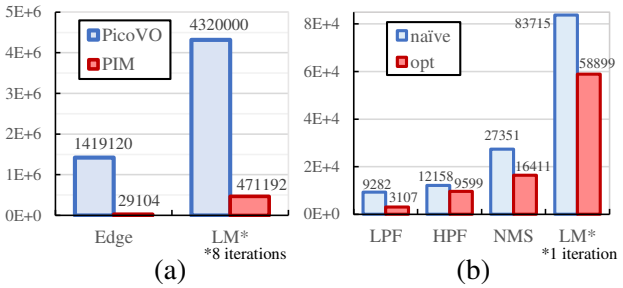

Figure 9: Computing cycles: (a) PicoVO and PIM EBVO; (b) naive and optimized implementations on PIM.

Table 1: RMSE Comparison of Relative Pose Error

	PicoVO		PIM EBVO	
Sequences	t.(m/s)	r.(°/s)	t.(m/s)	r.(°/s)
fr1_xyz	0.030	1.82	0.039	1.92
fr2_desk	0.020	0.69	0.019	0.64
fr3_st_ntex_far	0.028	0.77	0.030	0.86

Computing Logic: We implement the RTL design of the shifter, accumulator and register of 2560 bits, and utilize the Synopsys DC toolchain to synthesize our design in 90nm technology with 1.0V voltage and 216MHz frequency. The result shows that the chip area is 1.80E5 um^2, which is only 5.1% of the SRAM array, and the computing power is 44.6pJ per operation.

5.2 Accuracy

As shown in Tab.1, the tracking precision of the proposed PIM accelerated EBVO can compete with the state-of-the-art PicoVO implementation. Despite the minor round-off error caused by quantization and bit shifting during the feature warping and LM solver steps, the numerical precision of EBVO is very stable. Fig.8 exhibits the tracking and mapping results of two dataset sequences, which can prove the robustness of our PIM EBVO under scenarios with rich or poor features.

5.3 Computing Speed

Edge detection: The baseline PicoVO leverages a simplified detector called PicoEdge, and it takes up 1419120 cycles on board as shown in Fig.9-a. By contrast, the PIM acceleration only requires 3107 cycles for LPF, 9599 cycles for HPF, and 16411 cycles for NMS, summing up to 29117 cycles for the entire edge detection procedure. This reveals a 48x speedup over the PicoEdge detector. To show the performance improvement of the proposed PIM mapping scheme, we also compare it against a naive PIM implementation of those computing kernels, as depicted in Fig.9-b. This reveals an overall speedup ratio of 1.7x.

LM solver: PicoVO requires around 540000 cycles for each LM iteration, whereas on PIM, this overhead shrinks to 58899, which exhibits a 9x acceleration over the software implementation. Note that the speedup ratio is smaller than that of image processing, because the LM solver incurs a lot of 32-bit mul/div operations, which has higher complexity and 4x less throughput than the 8-bit image processing. We also compare the proposed computing pipeline with a naive PIM implementation (Fig.9-b), and the speedup ratio is 1.4x.

In the experiment, the LM solver converges within 8.1 iterations, and the overall speedup ratio over PicoVO is around 11x. This implies that the PIM accelerator can be clocked at a much lower frequency (~19MHz) than the MCU used in PicoVO, while still achieving the same performance.

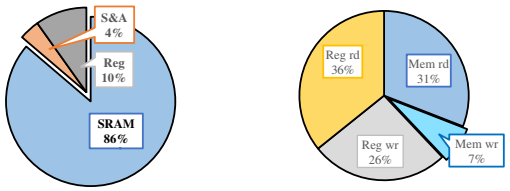

Figure 10: (a) Energy consumption of various PIM hardware components; (b) Memory access decomposition.

5.4 Energy Consumption

Without considering the I/O overhead, the baseline PicoVO takes 10.3mJ per frame on MCU. By contrast, the proposed PIM EBVO takes 0.495mJ per frame, which implies an improvement of 20.8x. To provide more insight, Fig.10-a decomposes the energy consumption of the PIM components: *SRAM*, *Shifter & Adder*, and *Tmp Reg*. It can be seen that 86% of power is consumed by *SRAM*, which is 7x more than the other PIM components. Thereby in our PIM EBVO implementation, we attempt to exploit the *Tmp Reg* as much as possible, so as to significantly reduce the SRAM write-back overhead, as shown in Fig.10-b. Using one *Tmp Reg* is a modest setup in this work, and we could use more registers to further improve the efficiency of both computation and power.

6 CONCLUSION

In this work we investigate the processing-in-SRAM acceleration for systematic optimization of the state-of-the-art EBVO. We analyze and simplify the key computation kernels required for the EBVO algorithm in the edge detection and pose estimation steps, and design specific computing patterns with PIM friendly memory layout for them accordingly. According to the calculation requirements of these kernels, we propose a bit-parallel SRAM-PIM architecture that can efficiently realize the high-precision arithmetical operations, while limiting the cost of both chip area and energy consumption. The experiments reveal that the PIM EBVO exhibits high tracking and mapping accuracy, and achieves an overall 11x acceleration of computing speed and 20x reduction of power consumption, compared to the baseline implementation that runs on embedded MCUs.

The proposed SRAM-PIM architecture has developed a general-purpose SIMD computing scheme for image processing and state estimation, and it may also benefit the integration of a broader range of applications such as CNN. In the future we plan to support more VO/vSLAM models, such as VIO and multiple camera sensor fusion, and realize our design in silicon.

Acknowledgments

This work is supported by the Strategic Priority Research Program of Chinese Academy of Sciences, Grant No. XDC05030201.

References

[1] Shaizeen Aga, Supreet Jeloka, Arun Subramaniyan, Satish Narayanasamy, David Blaauw, and Reetuparna Das. 2017. Compute caches. In *2017 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 481–492.

[2] Khalid Al-Hawaj, Olalekan Afuye, Shady Agwa, Alyssa Apsel, and Christopher Batten. 2020. Towards a reconfigurable bit-serial/bit-parallel vector accelerator using in-situ processing-in-sram. In *2020 IEEE International Symposium on Circuits and Systems (ISCAS)*. IEEE, 1–5.

[3] Mustafa Ali, Akhilesh Jaiswal, Sangamesh Kodge, Amogh Agrawal, Indranil Chakraborty, and Kaushik Roy. 2020. IMAC: In-memory multi-bit multiplication and ACcumulation in 6T SRAM array. *IEEE Transactions on Circuits and Systems I: Regular Papers* 67, 8 (2020), 2521–2531.

[4] Charles Eckert, Xiaowei Wang, Jingcheng Wang, Arun Subramaniyan, Ravi Iyer, Dennis Sylvester, David Blaaauw, and Reetuparna Das. 2018. Neural cache: Bit-serial in-cache acceleration of deep neural networks. In *2018 ACM/IEEE 45th Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 383–396.

[5] Jakob Engel, Thomas Schöps, and Daniel Cremers. 2014. LSD-SLAM: Large-scale direct monocular SLAM. In *European conference on computer vision*. Springer.

[6] Pedro F Felzenszwalb and Daniel P Huttenlocher. 2012. Distance transforms of sampled functions. *Theory of computing* 8, 1 (2012), 415–428.

[7] Christian Forster, Matia Pizzoli, and Davide Scaramuzza. 2014. SVO: Fast semi-direct monocular visual odometry. In *2014 IEEE international conference on robotics and automation (ICRA)*. IEEE, 15–22.

[8] Daichi Fujiki, Scott Mahlke, and Reetuparna Das. 2019. Duality cache for data parallel acceleration. In *Proceedings of the 46th International Symposium on Computer Architecture*. 397–410.

[9] Anders Grunnet-Jepsen, Michael Harville, Brian Fulkerson, Daniel Piro, Shirit Brook, and Jim Radford. [n.d.]. *Introduction to Intel® RealSense™ Visual SLAM and the T265 Tracking Camera*. https://dev.intelrealsense.com/docs/intel-realsensetm-visual-slam-and-the-t265-tracking-camera

[10] Xiaochen Guo, Engin Ipek, and Tolga Soyata. 2010. Resistive computation: Avoiding the power wall with low-leakage, STT-MRAM based computing. *ACM SIGARCH computer architecture news* 38, 3 (2010), 371–382.

[11] Yuquan He, Ying Wang, Cheng Liu, and Lei Zhang. 2021. PicoVO: A Lightweight RGB-D Visual Odometry Targeting Resource-Constrained IoT Devices. In *2021 IEEE International Conference on Robotics and Automation (ICRA)*. 5567–5573. https://doi.org/10.1109/ICRA48506.2021.9561285

[12] Jr. Henry S. Warren. [n.d.]. *Hacker's Delight: The Basics*. https://www.informit.com/articles/article.aspx?p=1959565{&}seqNum=19

[13] Mark Horowitz. 2014. 1.1 Computing's energy problem (and what we can do about it). In *2014 IEEE International Solid-State Circuits Conference Digest of Technical Papers (ISSCC)*. 10–14. https://doi.org/10.1109/ISSCC.2014.6757323

[14] Supreet Jeloka, Naveen Bharathwaj Akesh, Dennis Sylvester, and David Blaauw. 2016. A 28 nm configurable memory (TCAM/BCAM/SRAM) using push-rule 6T bit cell enabling logic-in-memory. *IEEE Journal of Solid-State Circuits* 51, 4 (2016), 1009–1021.

[15] Rainer Kümmerle, Giorgio Grisetti, Hauke Strasdat, Kurt Konolige, and Wolfram Burgard. 2011. g2o: A general framework for graph optimization. In *2011 IEEE International Conference on Robotics and Automation*. IEEE, 3607–3613.

[16] Kyeongho Lee, Jinho Jeong, Sungsoo Cheon, Woong Choi, and Jongsun Park. 2020. Bit parallel 6T SRAM in-memory computing with reconfigurable bit-precision. In *2020 57th ACM/IEEE Design Automation Conference (DAC)*. IEEE, 1–6.

[17] Runze Liu, Jianlei Yang, Yiran Chen, and Weisheng Zhao. 2019. eslam: An energy-efficient accelerator for real-time orb-slam on fpga platform. In *Proceedings of the 56th Annual Design Automation Conference 2019*. 1–6.

[18] Raul Mur-Artal and Juan D Tardós. 2017. Orb-slam2: An open-source slam system for monocular, stereo, and rgb-d cameras. *IEEE Transactions on Robotics* 33, 5 (2017), 1255–1262.

[19] Nicholas Nethercote and Julian Seward. 2007. Valgrind: a framework for heavy-weight dynamic binary instrumentation. *ACM Sigplan notices* 42, 6 (2007), 89–100.

[20] Fabian Schenk and Friedrich Fraundorfer. 2017. Robust edge-based visual odometry using machine-learned edges. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 1297–1304.

[21] Vivek Seshadri, Donghyuk Lee, Thomas Mullins, Hasan Hassan, Amirali Boroumand, Jeremie Kim, Michael A Kozuch, Onur Mutlu, Phillip B Gibbons, and Todd C Mowry. 2017. Ambit: In-memory accelerator for bulk bitwise operations using commodity DRAM technology. In *2017 50th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 273–287.

[22] Ali Shafiee, Anirban Nag, Naveen Muralimanohar, Rajeev Balasubramonian, John Paul Strachan, Miao Hu, R Stanley Williams, and Vivek Srikumar. 2016. ISAAC: A convolutional neural network accelerator with in-situ analog arithmetic in crossbars. *ACM SIGARCH Computer Architecture News* 44, 3 (2016), 14–26.

[23] Xin Si, Yung-Ning Tu, Wei-Hsing Huang, Jian-Wei Su, Pei-Jung Lu, Jing-Hong Wang, Ta-Wei Liu, Ssu-Yen Wu, Ruhui Liu, Yen-Chi Chou, et al. 2020. 15.5 a 28nm 64kb 6t sram computing-in-memory macro with 8b mac operation for ai edge chips. In *2020 IEEE International Solid-State Circuits Conference-(ISSCC)*. IEEE, 246–248.

[24] 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.

[25] Amr Suleiman, Zhengdong Zhang, Luca Carlone, Sertac Karaman, and Vivienne Sze. 2019. Navion: A 2-mw fully integrated real-time visual-inertial odometry accelerator for autonomous navigation of nano drones. *IEEE Journal of Solid-State Circuits* 54, 4 (2019), 1106–1119.

[26] Elene Terry. 2019. Silicon at the Heart of HoloLens 2. In *2019 IEEE Hot Chips 31 Symposium (HCS)*. IEEE Computer Society, 1–26.

[27] Yi Zhou, Hongdong Li, and Laurent Kneip. 2018. Canny-vo: Visual odometry with rgb-d cameras based on geometric 3-d–2-d edge alignment. *IEEE Transactions on Robotics* 35, 1 (2018), 184–199.