INCA: INterruptible CNN Accelerator for Multi-tasking in Embedded Robots

Jincheng Yu*†, Zhilin Xu*†, Shulin Zeng*†, Chao Yu*, Jiantao Qiu*†,
Chaoyang Shen*, Yuanfan Xu*, Guohao Dai*†, Yu Wang*† and Huazhong Yang*†
**Department of Electronic Engineering, Tsinghua University, Beijing, China*
†*Beijing National Research Center for Information Science and Technology (BNRist), Beijing, China*
yjc16@mails.tsinghua.edu.cn, yu-wang@mail.tsinghua.edu.cn

Abstract—**In recent years, Convolutional Neural Network (CNN) has been widely used in robotics, which has dramatically improved the perception and decision-making ability of robots. A series of CNN accelerators have been designed to implement energy-efficient CNN on embedded systems. However, despite the high energy efficiency on CNN accelerators, it is difficult for robotics developers to use it. Since the various functions on the robot are usually implemented independently by different developers, simultaneous access to the CNN accelerator by these multiple independent processes will result in hardware resources conflicts.**

To handle the above problem, we propose an INterruptible CNN Accelerator (INCA) to enable multi-tasking on CNN accelerators. In INCA, we propose a Virtual-Instruction-based interrupt method (VI method) to support multi-task on CNN accelerators. Based on INCA, we deploy the Distributed Simultaneously Localization and Mapping (DSLAM) on an embedded FPGA platform. We use CNN to implement two key components in DSLAM, Feature-point Extraction (FE) and Place Recognition (PR), so that they can both be accelerated on the same CNN accelerator. Experimental results show that, compared to the layer-by-layer interrupt method, our VI method reduces the interrupt respond latency to 1%.

I. Introduction

With the development of algorithms and hardware platforms, Convolutional Neural Network (CNN) has dramatically improved the perception and decision-making ability of unmanned platforms.

Distributed Simultaneously Localization and Mapping (DSLAM) is a basic task for many multi-robot applications, and is a hot topic in robotics. Two key modules consume most of the computation: Feature-point Extraction (FE) and Place Recognition (PR). FE provides the feature-points for the Visual Odometry (VO) to calculate the relative pose between two adjacent frames. PR generates compact image representation, which produces the candidate place recognition matches between different robots. Recent works use CNN to extract feature-points [1], [2] and generate the place representation code [3], [4]. The CNN-based feature-point extraction method, SuperPoint [1], achieves 10%-30% higher matching accuracy compared with the popular handcrafted extraction method, ORB [5]. The accuracy of the place recognition code from another CNN-based method, GeM [4], is also about 20% better than the handcrafted method, rootSIFT [6].

This work is supported by National Key Research and Development Program of China (No. 2017YFA0207600), National Natural Science Foundation of China (No. 61832007, U19B2019), Tsinghua EE Xilinx AI Research Fund, Beijing National Research Center for Information Science and Technology (BNRist), and Beijing Innovation Center for Future Chips. This work is also supported by China Postdoctoral Science Foundation (No. 2019M660641).

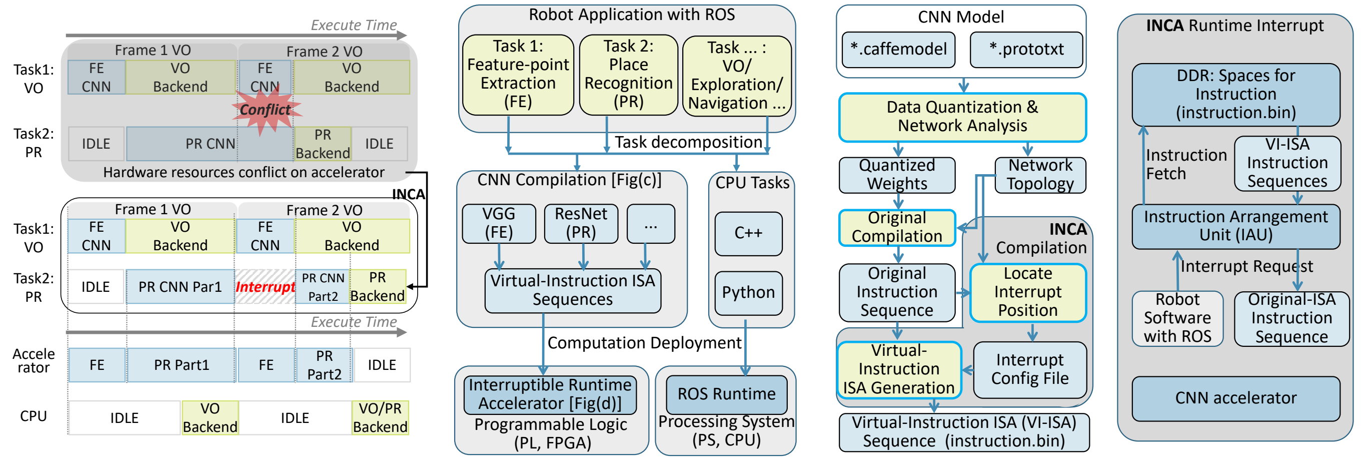
However, CNN is computation consuming. A single inference forward of the CNN-based GeM place recognition consumes 192G operations [4]. Thus, specific hardware architectures on FPGA [7]–[11] are designed to deploy CNN on the embedded system. With the help of network quantization and on-chip data reuse, the speed of CNN accelerators on embedded FPGA achieves 3TOP/s [11], which can support the real-time execution of CNN-based FE [1]. However, these CNN accelerators are designed and optimized to accelerate a single CNN. They can not automatically schedule two or more tasks simultaneously.

In order to facilitate robotic researchers to run different CNN tasks simultaneously on the FPGA accelerator, the accelerator should support the following features:

Multi-thread: Because different components in a robot are from different developers, thus, Robot Operating System (ROS) [12] is proposed as a middleware to fuse these independent components, and is widely used by robotic researchers. Each component is considered as an independent thread in ROS. Different threads should have independent access to the accelerator without knowing the status of others.

Finishing before deadline: In a robot, some tasks must be completed within the specified hard deadlines, such as FE. The moving robot's perception, including estimation of itself's location and the obstacles' position, is based on the feature-points. If FE is not completed before the deadline, the robot can not estimate the surrounding environment, causing collisions or even damage. Those critical tasks with a more stringent headline need to be performed prior to the non-critical tasks [13]. In DSLAM, the priority of feature-point extraction (FE) is higher than that of place recognition (PR). Because PR is only related to efficiency, yet FE ensures system safety.

To address the above challenges, we propose an INterruptible CNN Accelerator (INCA) for the rapid deployment of robot applications onto embedded FPGA. The workflow of INCA is illustrated in Figure 1(b). The first step is the task decomposition, which decomposes the computation of different tasks into CNN tasks and other CPU tasks. The


Execute Time
Task1: VO
Task2: PR
Frame 1 VO
Frame 2 VO
FE CNN
VO Backend
Conflict
IDLE
PR CNN
PR Backend
Hardware resources conflict on accelerator
INCA
PR CNN Par1
Interrupt
PR CNN Part2
Accelerator
FE
PR Part1
PR Part2
CPU
VO Backend
VO/PR Backend
Robot Application with ROS
Task 1: Feature-point Extraction (FE)
Task 2: Place Recognition (PR)
Task ... : VO/ Exploration/ Navigation ...
Task decomposition
CNN Compilation [Fig(c)]
VGG (FE)
ResNet (PR)
...
Virtual-Instruction ISA Sequences
CPU Tasks
C++
Python
Computation Deployment
Interruptible Runtime Accelerator [Fig(d)]
Programmable Logic (PL, FPGA)
ROS Runtime
Processing System (PS, CPU)
CNN Model
*.caffemodel
*.prototxt
Data Quantization & Network Analysis
Quantized Weights
Network Topology
Original Compilation
INCA Compilation
Original Instruction Sequence
Locate Interrupt Position
Virtual-Instruction ISA Generation
Interrupt Config File
Virtual-Instruction ISA (VI-ISA) Sequence (instruction.bin)
INCA Runtime Interrupt
DDR: Spaces for Instruction (instruction.bin)
Instruction Fetch
VI-ISA Instruction Sequences
Instruction Arrangement Unit (IAU)
Interrupt Request
Robot Software with ROS
Original-ISA Instruction Sequence
CNN accelerator

(a) INCA solves resources conflict. The CNN networks of different tasks time-multiplex the accelerator.

(b) INCA framework from robot application to hardware platform.

(c) At compilation, INCA adds virtual instructions to produce VI-ISA instruction sequences.

(d) At runtime, INCA translates the VI-ISA to original ISA executed on the accelerator.

Fig. 1. INCA framework. Fig(a) shows the diagram of scheduling feature-point based VO and place recognition (PR). The feature-point extraction (FE) and the PR are based on CNN and accelerated on the FPGA side. The VO backend to calculate the relative pose from feature-points and the PR backend to match the image representation are running at the CPU side. Fig(b),(c),(d) illustrate the workflow of INCA to deploy robot applications to embedded hardware platform with a CNN accelerator on it.

second step is to deploy the computation onto the FPGA. The CNN backbones of different tasks, such as the VGG model [14] in SupoerPoint feature-point extraction [1] and the ResNet101 model [15] in GeM place recognition [4], are compiled to the interruptible Virtual-Instruction Instruction Set Architecture (VI-ISA). With the help of the VI-ISA, the accelerator can be time-multiplexed with different tasks.

INCA facilitates robotic researchers to run different CNN tasks simultaneously on the FPGA with the following contributions:

- We propose a **virtual-instruction-based** interrupt method (VI method) to make the CNN accelerator support dynamic multi-task scheduling by priority. The method solves the hardware resources conflicts when accelerating different CNN tasks on ROS [12].
- We propose a CNN-based DSLAM system based on INCA. CNN-based methods for FE and PR are accelerated with FPGA on ROS. With the help of the unified interface in ROS, these CNN-based methods can be easily used by other developers in different applications.

II. Related Work

To accelerate CNN, some previous works design frameworks to generate a specific hardware architecture for a target CNN, based on RTL [9] or HLS [11]. These works need to reconfigure the FPGA to switch between different CNN models. The reconfiguration consumes seconds [16], which is unacceptable for the real-time system. Some other works design instruction-driven accelerators [7], [8], [10], [17], making rapid switching possible by providing different instruction sequences. However, the CNN tasks on previous instruction-driven CNN accelerators are not interruptible, resulting in the latency-sensitive high-priority task waiting for the low-priority task to finish. This inability of CNN accelerators to support multi-task makes it difficult for robotic researchers to use embedded FPGA.

III. INCA Frame work

Although ROS is becoming the fundamental software platform for robotics, the independence between different ROS tasks brings **hardware resources conflicts** to access the hardware accelerator. Figure 1(a) shows the time diagram of scheduling feature-point based visual odometry (VO) and Place Recognition in DSLAM system. The feature-point extraction (FE) and Place Recognition (PR) are implemented in CNN and deployed to the accelerator. In the native accelerator (the shadow part in Figure 1(a)), the threads of FE and PR may need to process CNN at the same time, and it leads to hardware resources conflicts.

Figure 1(a) also illustrates the idea of interrupt to schedule two CNN tasks. In the process of running a low-priority network (PR), the software may send an execution request for the high-priority task (FE). The interrupt enables the CNN accelerator to backup the running state of the low-priority PR network. Then the accelerator switches to the high-priority FE network. After the high-priority task (FE) completes, the low-priority task (PR) is restored to the accelerator and continues to execute.

Figure 1(c) details the INCA compilation step. Caffe [18] is a popular software framework for CNN, and the *.caffemodel/*.prototxt files define the network parameters and structure in Caffe. The previous deployment process, such as Angel-Eye [7] and DPU [17], quantizes the weights, and analyze the network topology. The original compiler translates the network topology and the quantization information into the original ISA sequence. INCA goes further than previous CNN

TABLE I
DESCRIPTION FOR THE INSTRUCTIONS

Type	Description	Backups	Recovery
LOAD_W	Load weights/bias from DDR to on chip weight buffer.	-	Weight / Inputdata
LOAD_D	Load input featuremaps from DDR to on chip weight buffer.	-	Weight / Inputdata
CALC_I	Calculate intermediate results for some output channels from partial input channels.	Intermediate data	Weight / Inputdata / intermediate data
CALC_F	Calculate the results for some output channels from all input channels.	Final results	Weight / Inputdata
SAVE	Save the results from on-chip data buffer to DDR.	-	Weight / Inputdata

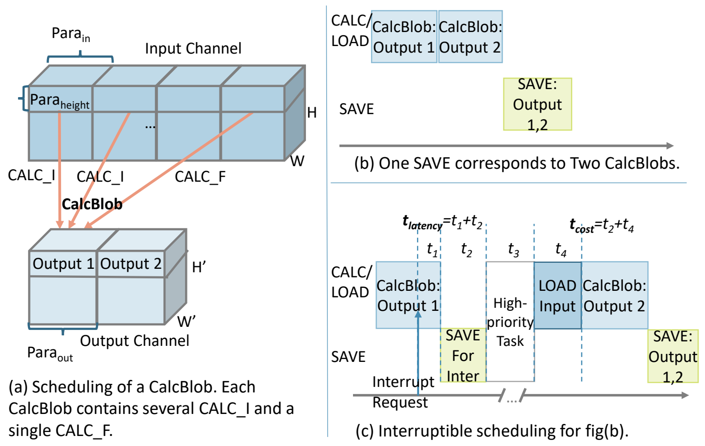


Fig. 2. Scheduling and interrupt for the instruction-driven accelerator.

compilers. It selects the optimized interrupt positions in the original instruction sequence, and adds virtual instructions at these positions to enable accelerator interrupt. After that, the original instruction sequence and the added virtual instructions are wrapped to the new interruptible VI-ISA. The wrapped VI-ISA instructions are dumped into a file (instruction.bin), and can be loaded into the instruction spaces on FPGA's DDR.

As illustrated in Figure 1(d), at runtime, an Instruction Arrangement Unit (IAU) in hardware listens to the interrupt request from ROS software, fetches the corresponding VI-ISA interruptible instructions and translates them to the original ISA executed on the CNN accelerator. Although we implement and evaluate INCA based on Angel-Eye [7], it can be applied to various instruction-based CNN accelerators.

IV. VIRTUAL-INSTRUCTION-BASED ACCELERATOR INTERRUPT

A. *Instruction Driven Accelerator*

There are three categories of instruction in the instruction-driven accelerator: LOAD (LOAD_W / LOAD_D), CALC (CALC_I / CALC_F), and SAVE [7], [8], [10]. The instruction description of each kind of instruction is listed in Table I.

Each CALC instruction, including CALC_I and CALC_F, processes the convolution according to the hardware parallelism with $Para_{height}$ lines from $Para_{in}$ input channels to $Para_{out}$ output channels. $Para_{height}$, $Para_{in}$, and $Para_{out}$ are the parallelism along the height, input channel and output channel dimensions, which is determined by the hardware and the original ISA. The convolution of the last $Para_{in}$ input channels is CALC_F, and the convolutions for the former input channels are CALC_I, as illustrated in Figure 2(a). The CALC_F and the CALC_I instructions for the same output channels, as well as the LOAD instructions for corresponding input featuremaps and weights, are considered as a **CalcBlob**.

B. *How To Interrupt: Virtual Instruction Inside Layer*

There are four stages to handle interrupt. For the instruction flow illustrated in Figure 2(b), the interrupt stages are shown in Figure 2(c), including: (1) Time for finishing the current operation, $t1$. (2) Time to backup, $t2$. (3) Time for the high-priority task, $t3$. (4) Time to restore the low-priority task ,$t4$. The latency to respond the interrupt is $t_{latency} = t_1 + t_2$. The extra cost for interrupt is $t_{cost} = t_2 + t_4$. There are different methods to implement interrupt in CNN accelerators.

CPU-Like. When an interrupt request occurs in CPU, CPU backs up all the on-chip registers to DDR. However, there are only tens of registers in CPU, and the volume of the backed-up data is less than 1 KB [19]. In CNN accelerators, there are several MB of on-chip caches [7], [10] for input featuremaps and weights. Thus, the extra data transfer increases both the interrupt response latency($t_{latency}$) and the additional cost (t_{cost}). CPU-Like interrupt needs a lot of extra memory space on DDR.

Layer-by-layer. Most accelerators run the CNN layer by layer [7], [10]. There is no extra data transfer for the accelerator to switch between different tasks after each layer, thus, $t_{cost} = 0$. However, the position of the interrupt request is irregular and unpredictable. When an interrupt occurs inside a CNN layer, the CNN accelerator needs to finish the whole layer before switching, which leads to the high response latency($t_{latency}$).

We propose the **virtual-instruction-based** method (VI method) to enable low-latency interrupt. Our VI method is interruptible inside each layer. Virtual instructions are some special instructions in the original instruction sequences. If an interrupt occurs, virtual instructions are executed to back up and restore the running state. If no interrupt occurs, the virtual instructions will not be executed. We add some virtual instructions to the original instruction sequence to enable the interrupt, which contain the vitrual SAVE (Vir_SAVE) and vitrual LOAD (Vir_LOAD) instructions. Vir_SAVE and Vir_LOAD are responsible for backing up and restoring on-chip caches respectively.

C. *Where To Interrupt: After CALC_F/SAVE*

We analyze the interrupt cost and select the positions of adding the virtual instructions. The backup/recovery data for different interrupt positions at each kind of instruction are listed in the Backup/Recovery columns of Table I.

When an interruption occurs at LOAD, the newly loaded data are immediately flushed when running the high-level CNN, leading to bandwidth waste.

Compared with CALC_I, when an interrupt occurs at CALC_F, there are no intermediate results. Although it is necessary to back up the unsaved final results immediately, these results will be stored in DDR through the subsequent normal non-virtual SAVE instruction. If the accelerator can record the interrupt status, we can modify the address and workload when executing subsequent normal non-virtual SAVE instruction. Thus, we can avoid the repetitive transmission of the final output results. The extra data transfer is only to recovery input data without any extra backup data, $t_{cost} = t_4$.

There is no data that need to be backed up when interrupt after SAVE. The overhead of interrupt after SAVE is also only to restore input data from DDR to the on-chip caches, $t_{cost} = t_4$.

In order to minimize the cost of interrupt, we make the CNN interruptible after the SAVE or CALC_F. This method only introduces extra data transfer to recovery input data without any extra backup data. Thus, $t_{cost} = t_4$, in our virtual-instruction-based interrupt.

Compared with layer-by-layer interrupt, our method, which is interruptible after CALC_F and SAVE, significantly reduces $t_{latency}$. In the worst case, the interrupt request occurs at the beginning of the layer. In this case, the accelerator will wait until finishing the whole layer. The wait time is t_{1_layer}:

$$t_{1\_layer} = \frac{Ch_{in} \times Ch_{out} \times H}{Para_{in} \times Para_{out} \times Para_{height}} \times t_{instr}(W)$$

Where $t_{instr}(W)$ is calculation time of a single CALC. The W of the input featuremaps is larger, the time of a single CALC is longer.

The worst wait of our VI method is t_{1_VI}:

$$t_{1\_VI} = \frac{Ch_{in} \times Para_{out} \times Para_{height}}{Para_{in} \times Para_{out} \times Para_{height}} \times t_{instr}(W)$$

Compared with the layer-by-layer method, the worst latency of our method is reduced to R_l.

$$R_l = \frac{t_{1\_VI}}{t_{1\_layer}} = \frac{Para_{out} \times Para_{height}}{Ch_{out} \times H} \tag{1}$$

The effect of latency reduction of the VI method is related to the number of output channels (Ch_{out}) and featuremap height (H). The larger the featuremaps output channels and the height, the better latency reduction result can be achieved.

Since usually $Ch_{in} \gg Para_{out}$, compared with the time of finishing current calculation (need to read data from all Ch_{in}), the time of backing up the final result (only save data for a $Para_{out}$) can be ignored. So that in both VI and layer-by-layer methods, the interrupt respond latency $t_{latency}$ is mainly determined by waiting for current calculation t_1. Experimental results of VI method in Section V-B include the backup time (t_2), yet the acceleration ratio is similar to the theoretical result of Equation (1).

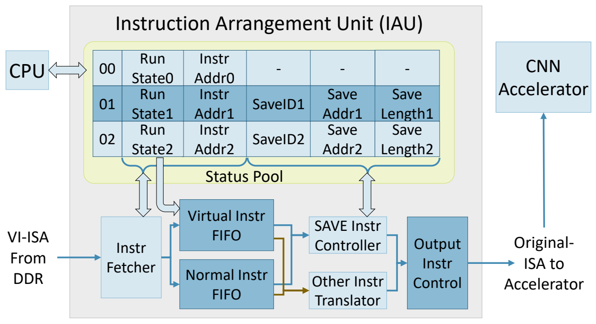


Fig. 3. Hardware architecture of IAU.

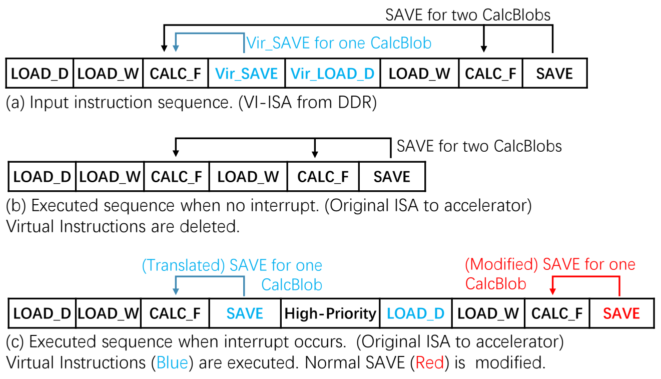

SAVE for two CalcBlobs
Vir_SAVE for one CalcBlob
LOAD_D
LOAD_W
CALC_F
Vir_SAVE
Vir_LOAD_D
LOAD_W
CALC_F
SAVE
(a) Input instruction sequence. (VI-ISA from DDR)
SAVE for two CalcBlobs
LOAD_D
LOAD_W
CALC_F
LOAD_W
CALC_F
SAVE
(b) Executed sequence when no interrupt. (Original ISA to accelerator)
Virtual Instructions are deleted.
(Translated) SAVE for one CalcBlob
(Modified) SAVE for one CalcBlob
LOAD_D
LOAD_W
CALC_F
SAVE
High-Priority
LOAD_D
LOAD_W
CALC_F
SAVE
(c) Executed sequence when interrupt occurs. (Original ISA to accelerator)
Virtual Instructions (Blue) are executed. Normal SAVE (Red) is modified.

Fig. 4. A simple example of our proposed virtual-instruction-based interrupt.

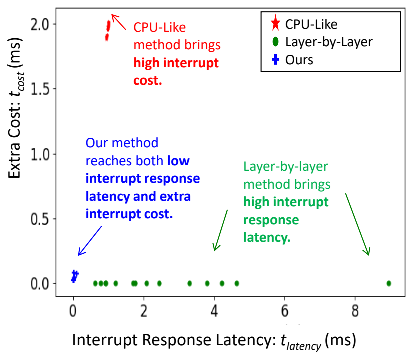
D. *Instruction Arrangement Unit (IAU)*

Instruction Arrangement Unit (IAU) is the hardware to handle the interrupts from the tasks with different priorities. IAU is shown in Figure 3, supporting 3 tasks with different priorities (priority 0,1,2). Task 0 has the highest priority and is not interruptible. *Status Pool* records the running status of the task at each priority. The *Instruction Fetcher* reads the VI-ISA instruction sequences from DDR according to the running state (*Run State*) and the DDR address of instructions (*Instr Addr*). *Virtual Instr FIFO* decides whether a virtual instruction needs to be executed according to the running state.

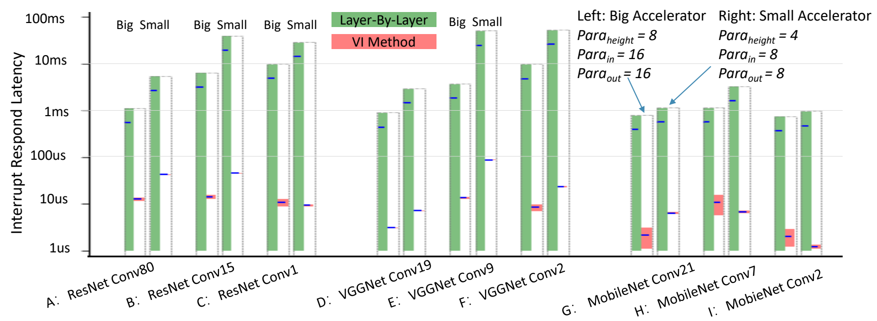
SAVE Instruction Controller writes the status of the executed virtual SAVE instructions to *Status Pool*, including ID of its corresponding normal non-virtual SAVE (*SaveID*), and the address (*Save Addr*) and length (*Save Length*) of the backed-up final output results. *SAVE Instruction Controller* also modifies the normal non-virtual SAVE instructions according to the recorded ID, Addr, Length to avoid duplicate data transfer for the final output results.

Instructions are translated from VI-ISA to the original ISA via *Instruction Translator* and *Output Instruction Controller*. The translated instructions are directly executed on the CNN accelerator. The CNN accelerator does not need to know the interrupt status, and only operates the instructions provided by IAU.

A simple example is given in Figure 4. Figure 4(a) is the instruction sequence from DDR with VI-ISA. The instructions are generated for the scheduling shown in Figure 2. The



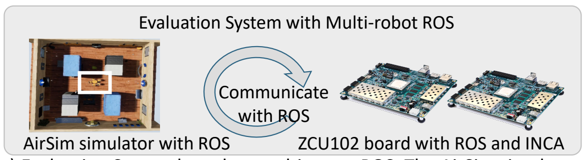
(a) Latency and cost comparison for different interrupt positions in PR task in DSLAM. Our method achieves both low latency and low cost.



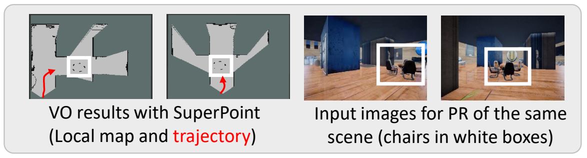
Layer	A	B	C	D	E	F	G	H	I
Network	ResNet101			VGGNet			MobileNet		
W x H	41 x 31	160 x 120	640 x 480	7 x 7	56 x 56	224 x 224	14 x 14	56 x 56	112 x 112
Chin x Chout	1024 x 256	128 x 128	3 x 64	512 x 512	256 x 256	64 x 128	512 x 512	256 x 256	32 x 32
kernel	1	3	7	3	3	3	1	1	3

(b) The latency of VI method is reduced to ~100 *us* under different layers (A-I) and different hardware parallelism (Big and Small). Performance of our method is stable, and the difference between best and the worst case small. The blue line (average) may overlap the red bar (the latency range of VI method) in some cases.

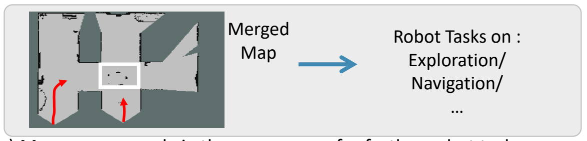
Fig. 5. Fig(a), comparison for interrupt CNN-based PR in DSLAM between different methods. Fig(b), latency comparison between layer-by-layer interrupt method and our virtual-instruction (VI) method.


Evaluation System with Multi-robot ROS
Communicate with ROS
AirSim simulator with ROS
ZCU102 board with ROS and INCA

(a) Evaluation System based on multi-agent ROS. The AirSim simulator runs on the server. The computation runs on ZCU102 boards.


VO results with SuperPoint (Local map and trajectory)
Input images for PR of the same scene (chairs in white boxes)

(b) The map and trajectory are generated by VO based on FE. The two input pictures are from the same scene are described and matched.



(c) Maps are merged via the same scene for further robot tasks.

Fig. 6. DSLAM: environment and results.

Vir_SAVE instruction is responsible for backing up executing status, i.e., saving the output results of the first CALC_F. The Vir_LOAD instruction restores the input featuremaps from DDR to on-chip memory. The Vir_LOAD. The SAVE instruction in Figure 4(a) saves all the output results for the two CALC_F. Figure 4(b) is the original ISA instructions translated by the IAU without interrupt. The virtual instructions (Vir_SAVE and Vir_LOAD) are skipped and discarded by the IAU. Thus the accelerator receives the original ISA sequence without backup/restore instructions. When an interrupt occurs at the first CalcBlob, Figure 4(c) illustrates the backup/recovery instructions (Blue) and the modified SAVE instruction (Red). Because the output results of the first CALC_F are stored to DDR with the first SAVE, which is translated from the Vir_SAVE in fig(a), the last SAVE instruction is modified to only store the output results of the second CALC_F.

V. Evaluation and Results

A. Experiment Setup

The hardware-in-loop evaluate environment is illustrated in Figure 6(a). There is a simulation server providing the simulation environment based on AirSim simulator [20], which provides the camera data for the two agents. Two Xilinx ZCU102 boards [21], with ZU9 MPSoC, are responsible for the calculation of each agent. SuperPoint [1] is used for extracting feature-points. GeM [4] is used for the PR module.

The CNN is calculated by the Angel-Eye CNN accelerator [7] on the FPGA side of ZU9 MPSoC, and other operations are on the CPU side. For a precise evaluation of the CNN running time, we record the clock cycles of the beginning and end of each instruction. The time of the interrupt response latency and the total cost in the following evaluation is calculated from the clock cycles and the clock frequency. The CNN accelerator and the IAU are running at 300MHz.

B. Evaluation of Virtual-Instruction Interrupt

In DSLAM, only the low-priority PR task is interruptible, and the interrupt position is unpredictable. GeM [4] is used to implement the PR module in the experiment. The CNN backbone of the GeM is ResNet101 [15], which contains 101 convolution layers. The input shape of the CNN is $480 \times 640 \times 3$. The parallelism of the Angel-Eye is $Para_{height} = 8$, $Para_{in} = 16$, $Para_{out} = 16$. We randomly sample 12 positions of the ResNet101 CNN backbone. The interrupt response latency and the extra time cost for different implementation of interrupt at the positions are listed in Figure 5(a). The CPU-like interrupt consumes the most extra cost (t_{cost}). Though the layer-by-layer interrupt consumes no extra time, the latency is much higher than our virtual-instruction-based interrupt. This is because the layer-by-layer interrupt needs to wait for the completion of a layer. The performance at the same interrupt position in our proposed virtual interrupt can interrupt inside a layer, with lower latency.

Furthermore, though the network structures differ between different CNNs, the convolutional layers, which are the basic component in CNN, are similar between different CNNs. INCA monitors the running status inside each layer, and the interrupt respond latency and extra cost are only relevant to the currently operating layer. We compare the interrupt respond latency of our VI method with layer-by-layer method at different layers from different networks, including ResNet [15], VGG [22], and MobileNet [23]. The results are illustrated in Figure 5(b). We evaluate our method on both the big accelerator with large hardware parallelism and small accelerator with small parallelism. In the ResNet and VGG, the average interrupt respond latency of the layer-by-layer method is ms to tens of ms, which makes the high priority task with hard deadline in the embedded system unable to be completed on time. With our VI method, the latency can be reduced to $\sim$100 us, so that the high priority task can be started immediately and completed on time. For the lightweight network (MobileNet), although the latency of the layer-by-layer method is $\sim$1 ms, we can still reduce the latency by 2-3 orders of magnitude with VI method. This result also matches the theoretical analysis in Equation (1).

C. *DSLAM with INCA*

The result of the DSLAM based on INCA is shown in Figure 6. The space in the AirSim [20] for the robots to explore is shown in Figure 6(a). It is a simple rectangle area with four different pillars, and some chairs at the center (in the white box). Figure 6(b) shows how PR works for map merging. The VO based on feature-points on each agent produces the local map and trajectory. When the PR threads find out a similar scene, and the maps and the trajectories are merged via the similar scene, as shown in Figure 6(c).

In this example, the CNN-based feature-point extraction (FE) and place recognition (PR) are both executed on the same Angel-Eye [7] accelerator. The frequency of the input camera is 20fps, and each input frame is fed to the FE, and the FE module would take up the accelerator. While the CPU process VO with the feature-points from FE, the accelerator can switch to process the low-priority PR task, as illustrated in Figure 1(a). Thus, the PR process one frame every 7$\sim$10 input frames. However, the scene between adjacent frames is similar, so it is not necessary to do place recognition for each input picture. Place recognition every 10 frames can meet the task requirements of DSLAM.

VI. Conclusion

In this paper, we propose an interruptible CNN accelerator and a deployment framework, INCA. With the help of the virtual-instruction-based interrupt method (VI method), the CNN accelerator can switch between different CNN tasks with low interrupt response latency and low extra cost. INCA currently focuses on interrupt support for single-core multi-tasking. We plan to investigate the multi-core multi-tasking for CNN accelerators as part of future work.

References

[1] D. DeTone, T. Malisiewicz, and A. Rabinovich, "Superpoint: Self-supervised interest point detection and description," in *CVPR Workshops*, 2018, pp. 224–236.

[2] K. M. Yi, E. Trulls, V. Lepetit, and P. Fua, "Lift: Learned invariant feature transform," in *ECCV*. Springer, 2016, pp. 467–483.

[3] R. Arandjelovic, P. Gronat, A. Torii, T. Pajdla, and J. Sivic, "Netvlad: Cnn architecture for weakly supervised place recognition," in *CVPR*, 2016, pp. 5297–5307.

[4] F. Radenović, G. Tolias, and O. Chum, "Fine-tuning cnn image retrieval with no human annotation," *IEEE transactions on pattern analysis and machine intelligence*, vol. 41, no. 7, pp. 1655–1668, 2018.

[5] R. Mur-Artal and J. D. Tards, "ORB-SLAM2: An Open-Source SLAM System for Monocular, Stereo, and RGB-D Cameras," *IEEE Transactions on Robotics*, vol. 33, pp. 1255–1262, 2016.

[6] H. Jégou and A. Zisserman, "Triangulation embedding and democratic aggregation for image search," in *CVPR*, June 2014, pp. 3310–3317.

[7] K. Guo, L. Sui, J. Qiu, J. Yu, J. Wang, S. Yao, S. Han, Y. Wang, and H. Yang, "Angel-eye: A complete design flow for mapping cnn onto embedded fpga," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 37, no. 1, pp. 35–47, 2017.

[8] J. Yu, G. Ge, Y. Hu, X. Ning, J. Qiu, K. Guo, Y. Wang, and H. Yang, "Instruction driven cross-layer cnn accelerator for fast detection on fpga," *ACM Transactions on Reconfigurable Technology and Systems (TRETS)*, vol. 11, no. 3, p. 22, 2018.

[9] H. Li, X. Fan, L. Jiao, W. Cao, X. Zhou, and L. Wang, "A high performance FPGA-based accelerator for large-scale convolutional neural networks," in *FPL*. IEEE, 2016, pp. 1–9.

[10] J. Qiu, J. Wang, S. Yao, K. Guo, B. Li, E. Zhou, J. Yu, T. Tang, N. Xu, S. Song *et al.*, "Going deeper with embedded fpga platform for convolutional neural network," in *FPGA*. ACM, 2016, pp. 26–35.

[11] L. Lu, Y. Liang, Q. Xiao, and S. Yan, "Evaluating Fast Algorithms for Convolutional Neural Networks on FPGAs," in *FCCM*, Apr. 2017, pp. 101–108.

[12] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Y. Ng, "Ros: an open-source robot operating system," in *ICRA workshop*, vol. 3, no. 3.2. Kobe, Japan, 2009, p. 5.

[13] R. Ramsauer, J. Kiszka, D. Lohmann, and W. Mauerer, "Look mum, no VM exits! (almost)," *CoRR*, vol. abs/1705.06932, 2017. [Online]. Available: http://arxiv.org/abs/1705.06932

[14] J. Kim, J. Kwon Lee, and K. Mu Lee, "Accurate image super-resolution using very deep convolutional networks," in *CVPR*, 2016, pp. 1646–1654.

[15] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *CVPR*, 2016, pp. 770–778.

[16] K. Papadimitriou, A. Dollas, and S. Hauck, "Performance of partial reconfiguration in fpga systems: A survey and a cost model," *Acm Transactions on Reconfigurable Technology & Systems*, vol. 4, no. 4, pp. 1–24, 2011.

[17] "DNNDK User Guide - Xilinx," 2019. [Online]. Available: https://www.xilinx.com/support/documentation/user_guides/ug1327-dnndk-user-guide.pdf

[18] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell, "Caffe: Convolutional architecture for fast feature embedding," *arXiv preprint arXiv:1408.5093*, 2014.

[19] S. B. Furber, *ARM system-on-chip architecture*. pearson Education, 2000.

[20] S. Shah, D. Dey, C. Lovett, and A. Kapoor, "Airsim: High-fidelity visual and physical simulation for autonomous vehicles," in *Field and service robotics*. Springer, 2018, pp. 621–635.

[21] "Xilinx Zynq UltraScale+ MPSoC ZCU102 Evaluation Kit," 2019. [Online]. Available: https://www.xilinx.com/products/boards-and-kits/ek-u1-zcu102-g.html

[22] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.

[23] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, "Mobilenets: Efficient convolutional neural networks for mobile vision applications," *arXiv preprint arXiv:1704.04861*, 2017.