Towards Cross-Platform Inference on Edge Devices with Emerging Neuromorphic Architecture

Shangyu Wu[1], Yi Wang[1], Amelie Chi Zhou[1], Rui Mao[1], Zili Shao[2], Tao Li[3]
1. The National Engineering Laboratory for Big Data System Computing Technology, Shenzhen University, Shenzhen, China
2. Department of Computer Science and Engineering, The Chinese University of Hong Kong, Hong Kong, China
3. Department of Electrical and Computer Engineering, University of Florida, Gainesville, FL, USA.
shangyuwu1006@gmail.com, {yiwang, chi.zhou, mao}@szu.edu.cn, shao@cse.cuhk.edu.hk, taoli@ece.ufl.edu

***Abstract*—Deep convolutional neural networks have become the mainstream solution for many artificial intelligence applications. However, they are still rarely deployed on mobile or edge devices due to the cost of a substantial amount of data movement among limited resources. The emerging processing-in-memory neuromorphic architecture offers a promising direction to accelerate the inference process. The key issue becomes how to effectively allocate the processing of inference between computing and storage resources on an edge device.**

This paper presents *Mobile-I*, a resource allocation scheme to accelerate the *I*nference process on *Mobile* or edge devices. Mobile-I targets at the emerging 3D neuromorphic architecture to reduce the processing latency among computing resources and fully utilize the limited on-chip storage resources. We formulate the target problem as a resource allocation problem and use a software-based solution to offer the cross-platform deployment across multiple mobile or edge devices. We conduct a set of experiments using realistic workloads that are generated from Intel Movidius neural compute stick. Experimental results show that Mobile-I can effectively reduce the processing latency and improve the utilization of computing resources with negligible overhead in comparison with representative schemes.

***Index Terms*—Edge computing, memory management, scheduling, neuromorphic architecture, inference**

I. Introduction

THE acceleration of deep learning applications on specific hardware resources normally involves two stages: training and inference. The training stage usually takes several hours or days, and it is deployed in high-end servers. The inference stage uses the previously trained parameters to classify, recognize, and process unknown inputs [1]. With more powerful and intelligent edge devices, there is a trend to perform inference stage of deep learning locally on mobile or edge devices. Although this trend is promising, moving inference on edge devices has to solve several critical issues. The key limiting factor is the cost of a substantial amount of data movement among constrained computing and storage resources.

Many emerging neuromorphic architectures (e.g., processing-in-memory) have been proposed to solve this challenge [2]. They aim to place computing resources inside or near storage to form the 3D stacked memory architecture. Since the target hardware infrastructure can obtain a set of deterministic processing procedures of deep learning applications, both deep learning applications and underlying edge devices can be formally modelled. The problem of performing inference on an edge device can be transformed into the problem of resource allocation on dedicated computing and storage resources. Specifically, a CNN application can be modelled as a synchronous data flow graph or a directed acyclic graph [3], [4]. Then the target problem could be effectively solved using task scheduling schemes.

In the previous work, some techniques solved task scheduling problem on shared resources with a performance guarantee [5], [6]. Other works studied task scheduling for data flow graphs [7], and they can obtain optimal or near-optimal task schedules. Although the task model in this paper also represents a CNN application as a data flow graph, the CNN application has many unique properties. The task scheduling for general data flow graphs may not be applicable to solve the scheduling problem for CNN applications on edge devices. Some previous studies have proposed FPGA (Field-Programmable Gate Array) and ASIC (Application-Specific Integrated Circuit) accelerators to accelerate the data processing of deep learning applications [8], [9]. Although these hardware-based neuromorphic platforms can effectively improve the processing procedure for specific applications, they are limited by the scalability of the hardware platform.

This paper presents *Mobile-I*, a resource allocation scheme to accelerate the *I*nference process on *Mobile* or edge devices. The objective is to minimize the processing latency of CNN applications and improve the cross-platform scalability for mobile or edge devices. Mobile-I first captures the properties of CNN applications and performs resource allocation on 3D neuromorphic architecture. The generated initial schedule can fully utilize computing resources. Mobile-I is compared with representative schemes in terms of processing latency, utilization of computing resource, and extra penalty for resource allocation. We conduct a set of experiments using realistic workloads that are generated from Intel neural compute stick [10]. Experimental results show that Mobile-I can achieve over 20% reductions in processing latency compared with previous studies. Moreover, Mobile-I can effectively utilize the

TABLE I
SOME TYPES OF PROCESSING OPERATIONS DOMINATE THE TOTAL PROCESSING TIME FOR REPRESENTATIVE CNN APPLICATIONS.

Neural Networks	**Conv2DBackpropFilter**	**Conv2DBackpropInput**	**Conv2D**	**BiasAddGrad**	**HistogramSummary**	**Others**
vgg_default	**40.83%**	**32.89%**	**12.03%**	4.91%	0.00%	9.34%
alexnet_mnist	**16.32%**	**28.67%**	**9.71%**	1.05%	**29.63%**	14.62%
cnn_sentence_classify	**32.65%**	**27.60%**	**12.45%**	3.14%	0.00%	24.16%
LeNet	**17.52%**	**15.63%**	4.29%	3.90%	**37.27%**	21.39%
resnet_cifar56	**49.01%**	**14.77%**	**5.50%**	**11.92%**	0.00%	18.80%
Average	31.27%	23.91%	8.80%	4.98%	13.38%	17.66%

hardware resources with negligible space and timing overhead.

The rest of this paper is organized as follows. Section II introduces system models used in the paper. Section III presents the proposed Mobile-I in detail. Section IV shows the experimental results. Finally, in Section V, we conclude this paper and discuss the future work.

II. SYSTEM MODELS AND MOTIVATION

A. System Architecture

The emerging neuromorphic architecture integrates both computing and storage resources in a single three-dimensional circuit [2]. It can be incorporated into the resource-constrained mobile or edge devices to accelerate the processing of deep learning applications. Figure 1 illustrates a typical system architecture, where several DRAM dies and a logic die are stacked. Each DRAM die is partitioned into 16 sub-dies. The sub-dies in the vertical direction form a vault, and each vault is individually controlled by the vault controller in the logic die [2]. Advanced 3D memory standards (e.g., Wide I/O-2 [11] and Hybrid Memory Cube (HMC) [12]) adopt similar system architectures.

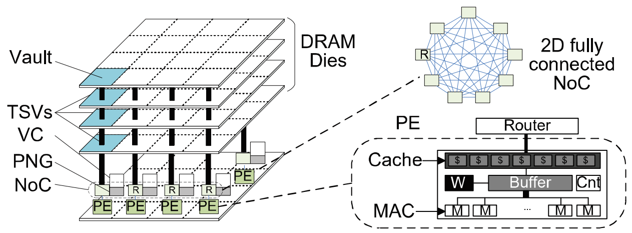


Fig. 1. The 3D-stacked neuromorphic architecture for CNN processing.

B. System Model

Convolutional neural network is inspired by the biological nervous system, and it is widely deployed in various deep learning systems. A CNN application has a standard structure with several functional layers, which can be categorized as *convolutional layers*, *pooling layers*, and *fully-connected layers*. Each layer can be further partitioned into a set of operations. Based on the data flow of the processing, a CNN application can be modelled as a directed acyclic graph or a synchronous data flow graph $G = (V, E, R)$, where $V = \{T_1, T_2, \dots, T_n\}$ represents a set of n tasks. Each task T_i denotes a computation operation, such as the fine-grained addition, inner product operation, or comparison operation. Each task T_i associates with an execution time e_i. Each edge $(T_i, T_j) \in E$ represents the data flow from one computation operation to another, and we use C_j^i to denote the corresponding data transfer. Similarly, e_j^i is the time cost or latency to transmit the required data from task T_i to T_j.

C. Motivation

The inference on edge devices should consider the properties of the target deep learning application. For a CNN application, we modelled it as a directed acyclic graph. Due to the nature of inference, the processing procedure for one CNN application does not affect the processing procedure for another CNN application. For example, an image processing application may receive several images, and the application will recognize the feature of each image. Since the training stage has already determined the parameters to inference, the inference stage can concurrently handle multiple images without causing data dependency issue. As a result, the processing for a group of CNN applications will share the same training parameters. This helps improve the utilization of hardware resources and reduces the processing latency.

We also observe that, for a CNN application, there are significant differences in processing time among different individual operations. We have obtained the processing time of several CNN benchmarks. These benchmarks are trained at GPU servers with TensorFlow. Table I illustrates the processing time in terms of the type of operations. For each benchmark, some types of operations dominate the total processing time, such as *Conv2DBackpropFilter* and *Conv2DBackpropInput*. For example, the operation *Conv2DBackpropFilter* takes 40.83%, 32.65%, and 49.01% processing time for benchmarks *VGG_default*, *CNN_sequence_classify*, and *resnet_cifar56*, respectively. The allocation of a CNN application should consider this variation in processing time.

Based on the model and analysis, we further formulate the problem as follows: *Given a directed acyclic graph* $G = (V, E, R)$ to model a CNN application, *the trained parameters for inference*, *and a mobile or edge device with 3D neuromorphic architecture*, *we aim to solve the following two problems:*

- *How to generate an initial schedule for computing tasks* $T_i, T_j \in V$, *such that the computing resources of the 3D neuromorphic architecture can be fully exploited?*
- *How to jointly reschedule both computing tasks* $T_i, T_j \in V$ *and corresponding data transfer* $C_j^i, (T_i, T_j) \in E$,

such that a schedule with the minimum processing latency can be generated?

III. Mobile-I: Inference for An Edge Device with Neuromorphic Architecture

A. Overview

Mobile-I aims to optimize the inference stage for mobile or edge devices. Mobile-I first abstracts the properties of a neural network and models it as a DAG. The inference stage will determine the allocation and running parameters of each operation for the target neural network. Specifically, Mobile-I creates an initial schedule in one period, with the objective of fully utilizing the computing resources. Based on this initial schedule, Mobile-I reallocates the execution of some operations in order to reduce the extra pre-processing overhead. For the last step, Mobile-I will generate the final schedule to optimize the allocation of data transfer on the PE's resource-constrained cache.

B. Generate an Initial Schedule

Mobile-I first generates an initial schedule to perform task allocation and task mapping. The initial schedule aims to provide a task schedule with guaranteed utilization ratio, and the utilization ratio U_p is defined as follows:

$$U_p = \frac{\sum_{T_i \in V} c_i \cdot X_p}{h \cdot c_p} \tag{1}$$

where X_p is the number of repeated execution for task set $\{T_1, \ldots, T_n\} \in V$. For an inference process, it enforces on the target neural network for X_{inf} times. For example, an image recognition application may handle a set of X_{inf} images, and each image is individually processed by the inference process. Then X_p is a subset of X_{inf} that can form a repeated execution for an inference process. c_i is the execution time of each computing task, h is the number of PEs, and c_p is the processing latency of these sets of tasks.

To improve the utilization ratio, the execution of X groups of task sets could be concurrently executed and utilized the empty time slots created by the execution of one single group of task sets. This involves in the mixed task schedule to prolong the execution of the task set. To preserve the data dependency relations between each pair of computing tasks T_i and T_j, (i.e., $(T_i, T_j) \in E$, and $\{T_i, T_j\} \in V$), task T_i has to be rescheduled into previous periods. The rescheduling of task T_i uses the retiming function $R(T_i)$. By retiming task T_i once, one period of task T_i is rescheduled into the newly added periods (called prologue). Prologue reflects the extra pre-processing latency. The retiming function has to ensure the data dependency, i.e.,

$$d_i + R(T_i) \times c_p + c_{i,j} \leq s_j + R(T_j) \times c_p, \qquad T_i \in V \tag{2}$$

where d_i is the finishing time of T_i and s_j is the starting (release) time of T_j. Then the processing latency of the schedule is

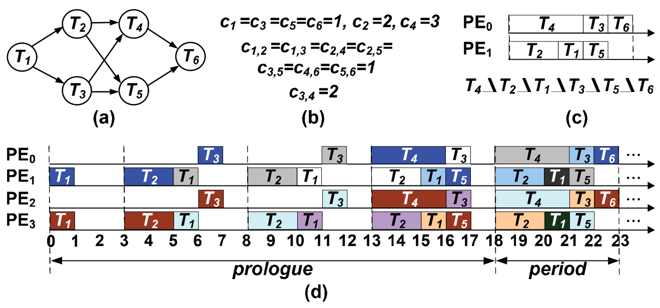
$$c_{sch} = (max\{|R(T_i)|\} + X) \times c_p, \qquad T_i \in V \tag{3}$$

Algorithm III.1 Generate an initial schedule for X_p groups of a period.

```
Require: A set of tasks $\{T_1, \ldots, T_n\} \in V$, $N_{PE}$ PEs, $h$ tasks that are concurrently executed within the same layer, PE's threshold utilization ratio $U_{lim}$, the maximum repeated times $X_{lim}$ of tasks in the same iteration.
Ensure: An initial schedule for $X_p$ groups of a period.
 1: Sort $\{T_1, \ldots, T_n\} \in V$ first by $c_i$, second by topology order.
 2: while $U_p < U_{lim}$ and $X_p < X_{lim}$ do
 3:     for each task $T_j \in V$ do
 4:         Assign $T_j$ to $PE_i$ with minimum $c_{p_i}$.
 5:         $c_{p_i} \leftarrow c_{p_i} + c_j$.
 6:     end for
 7:     $X_p \leftarrow X_p + 1$.
 8:     Update $U_p$.
 9: end while
10: if $X_p$ equals to $X_{lim}$ then
11:     Choose $X_p$ with the maximum $U_p$.
12: end if
```

Algorithm III.1 presents the basic steps to generate an initial schedule. It will first allocate the tasks with the longest execution time, and followed by the topology order. The task schedules with different groups are concatenated to form a new schedule in each period. The data dependency is changed from the intra-period dependency to the inter-period dependency, and it is guaranteed by the retiming operations. In this way, Mobile-I can generate a tight task schedule with the minimum processing latency within each period. Mobile-I defines the threshold utilization ratio U_{lim}. The utilization ratio U_p of the generated initial schedule in each period should be greater than U_{lim}. Mobile-I utilizes multi-issue scheduling to allocate tasks on multiple PEs of the neuromorphic architecture. We assume that h_G is the maximum degree of concurrency of the task graph G. Each issue follows the same task schedule. Then the time complexity of Algorithm III.1 is $O(X_{lim} \times n \times \log h)$.

Figure 2 illustrates an example for the generation of the initial schedule. We assume that the abstracted task graph consists of six tasks, the system contains 4 PEs, and U_{lim} is set as 90%. Figures 2(a) and 2(b) present the execution time and the data dependency. Since the current utilization ratio U_p=$(1 \times 9)/(2 \times 5) \geq 90\%$, the generated schedule does



(a)
$c_1=c_3=c_5=c_6=1$, $c_2=2$, $c_4=3$
$c_{1,2}=c_{1,3}=c_{2,4}=c_{2,5}=$
$c_{3,5}=c_{4,6}=c_{5,6}=1$
$c_{3,4}=2$
(b)
PE0
PE1
$T_4 \setminus T_2 \setminus T_1 \setminus T_3 \setminus T_5 \setminus T_6$
(c)
PE0
PE1
PE2
PE3
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23
prologue
period
(d)

Fig. 2. Generate an initial schedule for computing tasks. (a) The DAG to represent a neural network. (b) The execution time of each task. (c) The schedule in a period with allocation order. (d) The initial schedule with retiming to preserve data dependency.

Algorithm III.2 $UpdateNode(T_i, T_j)$.

```
Require: Tasks $T_i$ and $T_j$, moving interval $[l_i, r_i]$ of task $T_i$.
Ensure: Re-allocate task $T_i$ and update $[l_i, r_i]$.
 1: if the location of $T_i$ is not determined then
 2:     $Dis_{i,j} \leftarrow \max(0, s_j - d(i))$.
 3:     if $Dis_{i,j} \le c_{i,j}$ then
 4:         Get unselected node set $V_{unselect}$.
 5:         $d_i \leftarrow r_i - \sum_{T_k \in V_{unselect}} c_k$.
 6:         $R(i) \leftarrow \min(R(i), R(j))$.
 7:     else
 8:         $d_i \leftarrow r_i$.
 9:         $R(i) \leftarrow \min(R(i), \lfloor \frac{s_j + R(j) \times c_p - d_i - c_{i,j}}{c_p} \rfloor)$.
10:     end if
11:     Determine the location of $T_i$.
12: else
13:     $R(i) \leftarrow \min(R(i), \lfloor \frac{s_j + R(j) \times c_p - d_i - c_{i,j}}{c_p} \rfloor)$.
14: end if
```

not need to add an additional group of task sets, and each period is 5 time units. The generated initial schedule is shown in Figure 2(d). The maximum retiming operations is 4, and the prologue is 18 time units.

C. Reducing the Extra Processing Latency with Re-allocation of Tasks

Mobile-I further re-allocates the execution order in each period. For a pair of dependent tasks T_i and T_j, we first determine the location for successor task T_j, and use its location to further re-allocate the location for its predecessor task T_i. Algorithm III.2 presents the detailed steps to re-allocate predecessor task T_i. There are basically three cases: (1) $Dis_{i,j} \le c_{i,j}$ and T_i is ahead of T_j. Task T_i will be allocated to the end of the moving interval $[l_i, r_i]$, i.e., $d_i \leftarrow r_i$. (2) $Dis_{i,j} \le c_{i,j}$ and T_j is ahead of T_i. Task T_i in the previous period will be selected as the dependent task of T_j, and T_i should also be moved to the end of the moving interval. (3) $Dis_{i,j} > c_{i,j}$. For tasks within the moving interval of T_i, the undetermined tasks within this moving interval form a task set V_{move}. We can move these tasks to let the actual distance $Dis_{i,j}$ close to $c_{i,j}$. After determining the location of T_i, all these three cases should update the retiming value of T_i. The time complexity of Algorithm III.2 is $O(n)$.

Algorithm III.2 presents the methods to re-allocate tasks. It aims to provide timing interval for dependent tasks and let the irrelevant tasks to fill the time slot between dependent tasks. By doing this, the associated data transfer could be allocated and concurrently executed with these irrelevant tasks. An unsolved problem is to find the first successor task T_j and traverse all task sets from T_j.

We present the basic steps to select T_j. The starting point for traversing is the task with the longest processing latency, and we denote this kind of task as *critical task*. Mobile-I first creates the task set V_{c-set} for critical tasks. We rank tasks by the execution time, the longest execution time is c_{max}. If task T_i with execution time c_i is greater than $\alpha \cdot c_{max}$, where α is set as 0.8 in this paper. The rest task set belongs to $V_{uncheck}$. Figure 3 illustrates a run time example. Task T_4

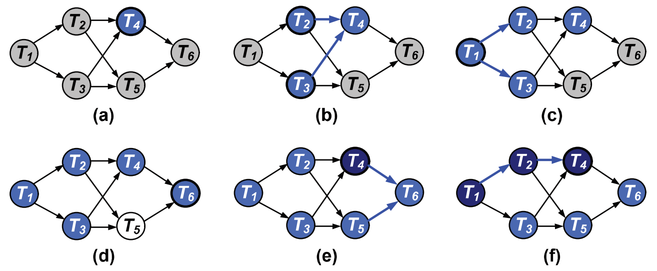


Fig. 3. Traverse the graph from critical tasks. (a) Select T_4 as the critical task. (b) Traverse predecessor tasks T_2 and T_3. (c) Traverse T_1 and finish the first round of traversing. (d) Select T_6 as the new critical task. (e) Traverse all predecessor tasks and record the status of T_4. (f) Update the retiming value and location of tasks from T_4.

is the critical task, and the traversing will start from T_4 and spread to its predecessor tasks T_2 and T_3. After traversing T_1, the first diffusion process is over. Then in Figure 3(d), a new critical task T_6 is selected. If its predecessor task has been visited (e.g., T_4), we will check whether the location of T_4 should be modified. Mobile-I only records this task, while not continuing traversing from task T_4. After all tasks have been traversed, Mobile-I begins to update all predecessor tasks starting from the recorded task with the largest topology order.

In previous steps, we generate a task schedule for computing tasks. The final task schedule is jointly affected by the allocation of computing tasks and their associated data transfer tasks. The allocation of computing tasks determines the relative data dependency across multiple periods. The last step is to allocate data transfer tasks with the objective of reducing the extra pre-processing latency for prologue. We record the iterative solutions with the subset of data transfer tasks. The final schedule for data transfer tasks could be obtained by recursively visiting the solution with the minimum prologue. Based on the model, Mobile-I can generate the final schedule,

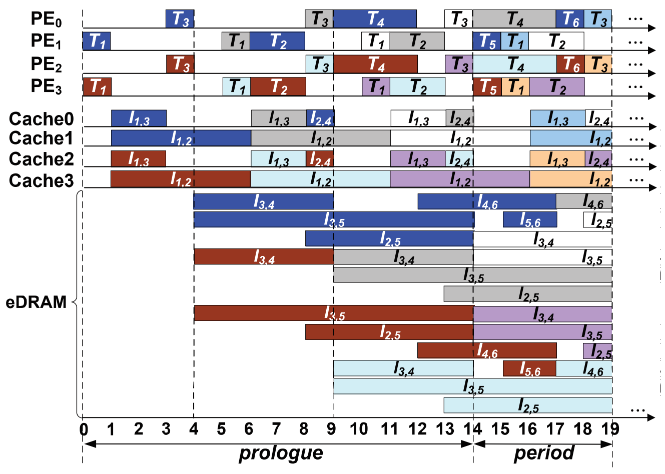


Fig. 4. The final schedule with high utilization ratio of PEs and reduced extra pre-processing for prologue.

TABLE II
THE PROCESSING LATENCY FOR MEMOLUTION [13], SPARTA [14], AND OUR MOBILE-I ON 16, 32, AND 64 PROCESSING ENGINES.

	# of	# of	16-PE			32-PE			64-PE		
Benchmarks	vertex	edge	SPARTA	Memolution	Mobile-I	SPARTA	Memolution	Mobile-I	SPARTA	Memolution	Mobile-I
MobileNetSSD	84	106	57.86	34.63	30.38	32.17	19.49	15.28	17.71	11.92	7.72
AgeNet	15	15	51.07	17.49	17.09	28.38	9.23	8.58	15.02	5.10	4.33
AlexNet	26	30	108.75	37.57	35.11	60.45	20.17	17.76	31.99	11.47	9.11
GENDERNET	15	15	50.31	17.24	16.97	27.96	9.10	8.52	14.80	5.03	4.30
GoogLeNet	78	159	83.40	39.28	36.58	45.06	21.18	18.46	23.52	12.13	9.40
Inception_ResNet	294	410	746.00	521.57	295.36	414.09	382.29	156.39	218.87	312.65	86.90
Inception_v1	75	155	84.50	36.92	34.51	46.09	19.82	17.43	23.99	11.27	8.88
Inception_v2	87	184	96.42	50.71	47.18	51.95	27.35	23.84	26.94	15.66	12.17
SqueezeNet	34	42	50.18	19.81	18.73	27.89	10.59	9.50	14.76	5.98	4.90
Tiny-YOLO	18	18	133.82	47.33	44.30	74.38	25.31	22.33	39.36	14.29	11.37

which is shown in Figure 4. Compared to the initial schedule (as shown in Figure 2), the final schedule further reduces the extra pre-processing latency by one period. For the inference process that needs to handle several hundreds of task sets, this newly added prologue can improve the utilization ratio of PEs and reduce the unnecessary extra processing latency caused by critical tasks.

IV. EVALUATION

A. Experimental Setup

We conducted experiments using a set of benchmarks from real-life CNN applications. The widely used open-source deep learning framework TensorFlow [15] is used to train neural network models and abstract graph representations. We utilized Intel Neural Compute Stick (NCS) [10] to track the timing parameters, data dependency relations, and transfer rate for each event. Intel NCS is a tiny fanless deep learning device to learn AI programming at the edge. The directed acyclic graphs are abstracted from TensorBoard and log files of TensorFlow to determine the functionality of the deep learning applications.

Mobile-I is compared with Memolution [13] and SPARTA [14]. Memolution presents a task allocation scheme for CNNs. SPARTA is a throughput-aware task scheduling scheme for multi-core system architecture. Therefore, these schemes are selected for comparison. The neuromorphic architecture is implemented based on the extension of the model in Neurocube [2]. The trained neural networks are deployed inference on target neuromorphic architecture.

B. Results and Discussion

1) Processing Latency: Table II presents the processing latency of Memolution [13], SPARTA [14], and the proposed Mobile-I under 16, 32, and 64 PEs. From the experimental results, Mobile-I can reduce the processing latency by 17.54% and 62.88% compared to Memolution and SPARTA, respectively. This is mainly due to joint optimization of the initial schedule and the re-order of critical tasks. SPARTA does not change data dependency relationships across multiple periods. Even though it can effectively allocate tasks with reduced processing latency, the data transfer will cost extra delay. Memolution adopts retiming technique to reschedule tasks. However, it does not consider the timing differences among multiple tasks. Therefore, the results for processing latency of their schemes are longer than our scheme.

2) Extra Latency for Pre-processing: Both Memolution and the proposed Mobile-I adopt retiming operations, and we report the normalized extra latency. From the experimental results in Figure 5, Mobile-I can significantly reduce the extra latency for pre-processing. The time cost is only 11.40% on average compared to Memolution. The reduction of the extra latency for pre-processing leads to the improvement of the processing efficiency. Mobile-I captures the unique properties of neural network applications and allocates the critical tasks with the highest priority. This optimization strategy effectively prevents the waste of time due to the allocation of critical tasks.

3) Utilization Ratio of PEs: Figure 6 illustrates the utilization ratio of PEs for Memolution, SPARTA, and our proposed Mobile-I. In Mobile-I, the threshold utilization ratio is set as 92%. As expected, Mobile-I achieves the highest utilization ratio (96.14% on average). The utilization ratios for Memolution and SPARTA are 79.75% and 35.72%, respectively. SPARTA has to maintain the data dependency within the same period, so it cannot overlap the execution of computing tasks with other data transfer tasks. Although Memolution can effectively improve its utilization ratio, its schedule does not handle critical tasks. The experimental results of extra latency for pre-processing also prove that some sets of the tasks have to be processed in the pre-processing procedure, which leads to the low utilization ratio of PEs. Mobile-I can effectively utilize both computing and storage resources, and they can

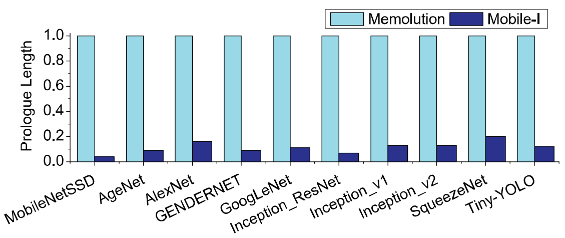


Fig. 5. The prologue time for Memolution [13] and our Mobile-I on 16, 32, and 64 processing engines.

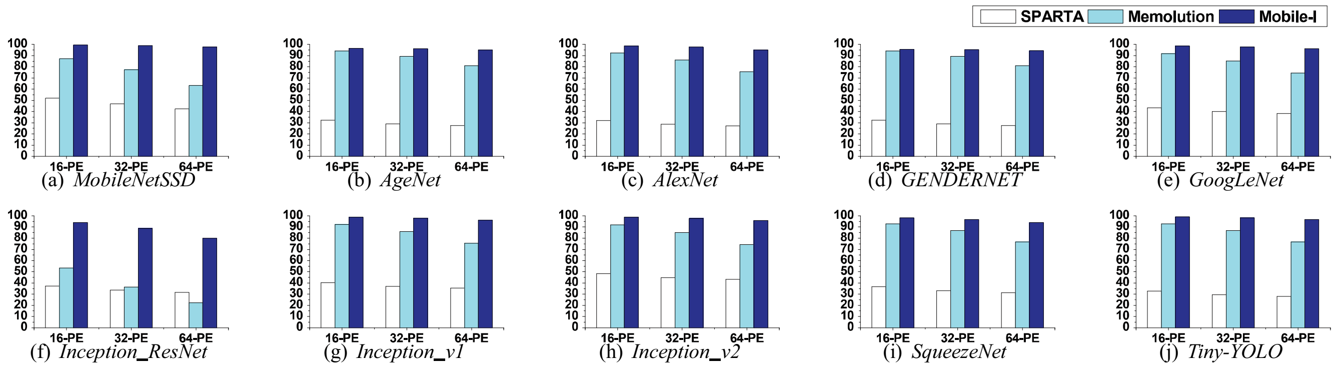


Fig. 6. The utilization ratio of processing engines for Memolution [13], SPARTA [14], and our Mobile-I on 16, 32, and 64 processing engines.

provide services with low latency.

V. Conclusion

This paper presents a task scheduling technique called *Mobile-I* to accelerate deep learning inference in edge devices. We jointly optimize both computing and storage resources such that the processing delay can be effectively reduced and cross-platform scalability can be improved. Experimental results show that the proposed Mobile-I can significantly reduce the processing delay and effectively utilize the limited resources on edge devices compared to the previous studies. In the future, we plan to investigate the use of our technique on other emerging neuromorphic architectures and propose a general scheduling framework that can be applied to different system architectures.

Acknowledgment

This work was supported in part by the National Natural Science Foundation of China (61502309 and 61802260), in part by the Guangdong Natural Science Foundation (2016A030313045, 2018A030310440, and 2017B030314073), in part by the Shenzhen Science and Technology Foundation (JCYJ20170817100300603, JCYJ20170816093943197, and J-CYJ20170302153955969), in part by the Research Grants Council of the Hong Kong Special Administrative Region, China (GRF 15222315, GRF 15273616, GRF 15206617, and GRF 15224918), in part by Direct Grant for Research, The Chinese University of Hong Kong (Project No. 4055096), in part by the Guangdong Pre-national Project (2014GKXM054), and in part by the Natural Science Foundation of SZU (803/000027060147). Yi Wang is the corresponding author.

References

[1] NVIDIA Corporation, "GPU-based deep learning inference: A performance and power analysis," *https://www.nvidia.com/content/tegra/embedded-systems/pdf/jetson_tx1_whitepaper.pdf*, 2018.

[2] D. Kim, J. Kung, S. Chai, S. Yalamanchili, and S. Mukhopadhyay, "Neurocube: A programmable digital neuromorphic architecture with high-density 3D memory," in *2016 ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA)*, June 2016, pp. 380–392.

[3] Y. LeCun, K. Kavukcuoglu, and C. Farabet, "Convolutional networks and applications in vision," in *Proceedings of 2010 IEEE International Symposium on Circuits and Systems (ISCAS)*, May 2010, pp. 253–256.

[4] Y. H. Chen, T. Krishna, J. S. Emer, and V. Sze, "Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks," *IEEE Journal of Solid-State Circuits*, vol. 52, no. 1, pp. 127–138, 2017.

[5] J. Han, X. Tao, D. Zhu, and L. Yang, "Resource sharing in multicore mixed-criticality systems: Utilization bound and blocking overhead," *IEEE Transactions on Parallel and Distributed Systems*, vol. 28, no. 12, pp. 3626–3641, 2017.

[6] J. Huang, R. Li, J. An, D. Ntalasha, F. Yang, and K. Li, "Energy-efficient resource utilization for heterogeneous embedded computing systems," *IEEE Transactions on Computers*, vol. 66, no. 9, pp. 1518–1531, Sept 2017.

[7] M. H. Foroozannejad, M. Hashemi, T. L. Hodges, and S. Ghiasi, "Look into details: The benefits of fine-grain streaming buffer analysis," in *Proceedings of the ACM SIGPLAN/SIGBED 2010 Conference on Languages, Compilers, and Tools for Embedded Systems (LCTES)*, 2010, pp. 27–36.

[8] C. Zhang, P. Li, G. Sun, Y. Guan, B. Xiao, and J. Cong, "Optimizing FPGA-based accelerator design for deep convolutional neural networks," in *Proceedings of the 2015 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays (FPGA)*, 2015, pp. 161–170.

[9] T. Chen, Z. Du, N. Sun, J. Wang, C. Wu, Y. Chen, and O. Temam, "DianNao: A small-footprint high-throughput accelerator for ubiquitous machine-learning," in *Proceedings of the 19th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2014, pp. 269–284.

[10] Intel Movidius Neural Compute Stick, "https://developer.movidius.com/," 2018.

[11] JEDEC Wide I/O-2 Standard, "https://www.jedec.org/standards-documents/docs/jesd229-2," 2017.

[12] J. T. Pawlowski, "Hybrid memory cube (HMC)," in *2011 IEEE Hot Chips 23 Symposium (HCS)*, Aug 2011, pp. 1–24.

[13] Y. Wang, M. Zhang, and J. Yang, "Towards memory-efficient processing-in-memory architecture for convolutional neural networks," in *Proceedings of the 18th ACM SIGPLAN/SIGBED Conference on Languages, Compilers, and Tools for Embedded Systems (LCTES)*, 2017, pp. 81–90.

[14] B. Donyanavard, T. Mück, S. Sarma, and N. Dutt, "Sparta: Runtime task allocation for energy efficient heterogeneous many-cores," in *Proceedings of the Eleventh IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis (CODES)*, 2016, pp. 27:1–27:10.

[15] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, M. Kudlur, J. Levenberg, R. Monga, S. Moore, D. G. Murray, B. Steiner, P. Tucker, V. Vasudevan, P. Warden, M. Wicke, Y. Yu, and X. Zheng, "Tensorflow: A system for large-scale machine learning," in *Proceedings of the 12th USENIX Conference on Operating Systems Design and Implementation (OSDI)*, 2016, pp. 265–283.