Exploiting Parallelism for CNN Applications on 3D Stacked Processing-In-Memory Architecture

Yi Wang, Weixuan Chen, Jing Yang, Tao Li

Abstract—Deep convolutional neural networks (CNNs) are widely adopted in intelligent systems with unprecedented accuracy but at the cost of a substantial amount of data movement. Although the emerging processing-in-memory (PIM) architecture seeks to minimize data movement by placing memory near processing elements, memory is still the major bottleneck in the entire system. Current PIM architecture can only provide 100-300 KB cache capacity to store the intermediate processing results and the filter weights. However, state-of-the-art CNN applications require over hundreds of megabytes for concurrent processing of convolutions. How to jointly explore the computation capability of the PIM architecture and the highly parallel property of neural networks remains a critical issue.
This paper presents *Para-Net*, that exploits *Para*llelism for deterministic convolutional neural *Net*works on the PIM architecture. Para-Net achieves data-level parallelism for convolutions by fully utilizing the on-chip processing engine (PE) in PIM. The objective is to capture the characteristics of neural networks and present a hardware-independent design to jointly optimize the scheduling of both intermediate results and computation tasks. We formulate this data allocation problem as a dynamic programming model and obtain an optimal solution. To demonstrate the viability of the proposed Para-Net, we conduct a set of experiments using a variety of realistic CNN applications. The graph abstractions are obtained from deep learning framework Caffe. Experimental results show that Para-Net can significantly reduce processing time and improve cache efficiency compared to representative schemes.

Index Terms—Near-data processing, neuromorphic computing, scheduling, memory management, parallel computing.

1 Introduction

Convolutional neural networks (CNNs) are widely applied in many deep learning applications such as speech recognition, natural language processing, and image processing. Computational efficiency is a primary concern in such deep learning systems, where hundreds of filters and channels in the high-dimensional convolutions have to be simultaneously processed. Current state-of-the-art CNNs require several hundreds of megabytes for filter weight storage and 30K-600K operations per input pixel [2], [3]. In such networks, convolutions normally take up about 90% of the CNN operations, and they dominate execution time [4]. The expensive data movement for convolutions poses throughput challenges to the underlying computing and storage hardware.

Processing-in-memory (PIM) is a promising solution to address the data movement challenges [5]. It seeks to moving computation inside or near memory. Recent progress in PIM presents 3D-stacked memory architectures to place processing engine (PE) close to the data in memory, thereby reducing the expensive off-chip fetching operations. This new architecture allows the stacked memory to serve both computation and memory functions. On the other hand, neural networks are data intensive and highly parallelizable, and there are many opportunities to exploit different levels of parallelism. For neural networks running on such PIM architectures, it is possible to fully exploit the parallelism of convolutions. However, a significant amount of intermediate processing results (i.e., partial sums) are generated by the parallel of convolutions, which require fairly large additional memory footprint to hold the untreated intermediate data.

This intermediate data impacts throughput, the fundamental attribute of a computing system. For data-intensive CNN applications, the overhead for throughput is no longer dominated by the computation of convolutions but instead by the handling of large data volume for the intermediate data and the cost of moving the intermediate data to the required processing engine. Minimizing the movement of intermediate data can significantly reduce the overall memory usage, which would be of great value in the resource constrained PIM architecture.

This paper presents *Para-Net*, a novel task-level data allocation framework to exploit parallelism for convolutional connections in the PIM architecture. Para-Net jointly reallocates both convolutions and intermediate data. The objective is to generate an optimal assignment with the maximum application throughput while minimizing the overall off-chip fetching operations. Para-Net adopts a two-stage framework to determine the computation of convolutions and the associated intermediate processing results. It captures the highly parallel feature of convolutions and lets a number of convolutions reschedule into earlier iterations. After changing intra-iteration data dependencies into inter-iteration data dependencies, the processing engine can be fully utilized and the throughput for convolutions can be effectively improved.

- *This version is a revised version. A preliminary version of this work appears in the Proceedings of the 54th ACM/IEEE Design Automation Conference (DAC 2017) [1].*
- *Y. Wang and W. Chen are with the College of Computer Science and Software Engineering, Shenzhen University, Shenzhen, 518060, China. J. Yang is with Experimental and Innovation Practice Center, Harbin Institute of Technology, Shenzhen, 518055, China. T. Li is with Department of Electrical and Computer Engineering, University of Florida, Gainesville, FL, USA.*

In the first stage, Para-Net generates an initial schedule for convolutions. This schedule considers both the utilization of PEs and the concurrent execution of convolutions. In the second stage, Para-Net transformed the problem into the minimization of the overall off-chip fetching penalty for intermediate data. We first perform analysis and theoretically obtain the upper bound of the times needed to reallocate each intermediate processing result. Based on this analysis, we formulate the problem as a dynamic programming model and obtain an optimal solution.

We evaluate the proposed technique with a set of benchmarks from real-life convolutional applications. The graph abstractions are obtained from the deep learning framework Caffe [6]. We compare Para-Net with the baseline scheme [7] and our previous work Para-CONV [1] in terms of throughput, prologue time, utilization ratio of PEs, cache efficiency, and time cost for program execution. Experimental results show that, Para-Net can achieve a significant reduction in execution time and effectively utilize the PIM architecture with the minimum overall data movement penalty.

The main contributions of this paper are:

- For the allocation of convolutions, a task scheduling framework is proposed to abstract the characteristics of CNN applications and generate a schedule with the efficient utilization of processing engines.
- For the allocation of intermediate processing results, the schedulability analysis is performed to obtain a tight upper bound to reschedule convolutions, and a dynamic programming model is presented to get an optimal allocation.
- The proposed scheduling framework is compared with representative works using a set of real-world CNN applications.

The rest of this paper is organized as follows. Section 2 introduces models and concepts used in this paper. Section 3 presents our proposed dynamic programming model in detail. Section 4 illustrates experimental results. Section 5 presents the literature review. Section 6 concludes the paper and discusses future work.

2 System Models and Concepts

2.1 System Model with 3D PIM Architecture

Advanced PIM architecture integrates a number of DRAM or embedded DRAM (eDRAM) arrays and a number of processing engines (PEs) in a 3D-stacked memory architecture [8]. Figure 1 illustrates the general 3D system architecture of CNN processing. This novel in-memory neuromorphic processing architecture places computation units close to the data in memory to further improve throughput [9]. Each PE integrates a PE FIFO (pFIFO), an ALU datapath, a register file (RF), and a data cache to store the intermediate CNN processing results. An input/output FIFO (iFIFO/oFIFO) is used to communicate the traffic among multiple PEs. Multiple PEs can concurrently communicate with multiple DRAM vaults through high-speed Through Silicon Vias (TSVs).

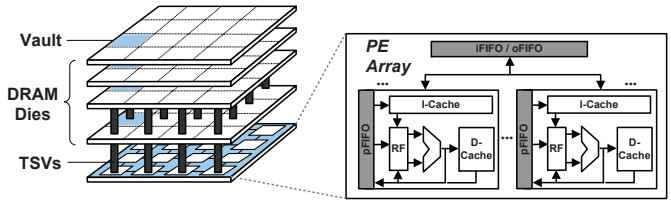


Fig. 1. The system architecture of the accelerator system for CNN processing.

2.2 Application Model for CNN

Convolutional neural network is designed to recognize visual patterns directly from pixel images with minimal preprocessing [10]. As shown in Figure 2(a), a CNN has a standard structure with multiple stacked convolutional layers, pooling layers, and one or more fully-connected layers. The primary computation of CNN is in the convolutional layers. Each convolutional layer applies several three-dimensional filters over a three dimensional input [4]. It is an inner product calculation, where pairwise multiplications are performed among the input elements (or neurons) and the filter weights (or synapses). The obtained products are further reduced into a single output neuron using addition. The pooling layer typically involves a simple maximum or average operation on a small set of input numbers. The fully-connected layer calculates inner products of its inputs and corresponding weights. It can be treated as a special kind of convolutional layers.

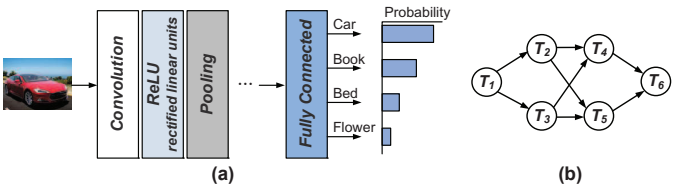


Fig. 2. (a) The standard procedure for CNN with multiple stacked layers. (b) A CNN can be modelled as a directed acyclic graph.

A CNN can be represented by a directed acyclic graph [11]. A graph $G = (V, E, P, R)$ is a weighted graph. $V = \{T_1, T_2, \ldots, T_n\}$ is the vertex set, and each vertex represents a convolution or pooling operation. $E \subseteq V \times V$ is the edge set, and each directed edge, $(T_i, T_j) \in E$ $(T_i, T_j \in V)$, represents the intermediate processing results generated from vertex T_i and requested by vertex T_j. For each directed edge $(T_i, T_j) \in E$, an intermediate processing result $I_{i,j}$ denotes the corresponding data transfer from a convolution operation T_i to a convolution operation T_j. Each graph represents the processing procedure of one single pixel image. For a CNN application with multiple visual patterns, it can be modelled as periodic dependent tasks.

A convolutional layer applies filters on the input feature maps to extract embedded visual characteristics and generate the output feature maps. Weight sharing is a unique property in convolutional layers, and a small amount of the input data (i.e, filter weight) can be shared across many operations. The number of convolutional sharing is bounded by the width of filter and output feature maps [12], [13]. Therefore, a CNN application can be further modelled as a

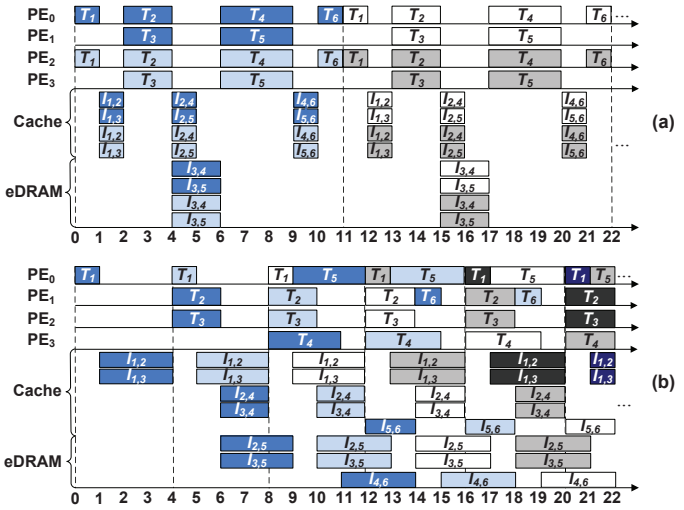


Fig. 3. (a) Intermediate processing results cause delay of convolution execution. (b) Fully exploiting parallelism needs joint optimization of convolution operations and intermediate processing results.

periodically executed dataflow with deterministic convolutional connections.

Let p be the period of each convolution operation T_i and that of each associated intermediate processing result $I_{i,j}$. A convolution operation T_i is associated with three tuples $T_i(s_i, c_i, d_i)$, where s_i is the start time, c_i is the execution time, and d_i is the deadline of T_i. For T_i in the ℓ-th iteration, T_i^ℓ, three tuples become $T_i^\ell(s_i^\ell, c_i^\ell, d_i^\ell)$, where $s_i^\ell = s_i + (\ell - 1) \cdot p$, $c_i^\ell = c_i$, and $d_i^\ell = d_i + (\ell - 1) \cdot p$, $\ell \geq 1$. Similarly, intermediate processing result $I_{i,j}$ in the ℓ-th iteration can be expressed by $I_{i,j}^\ell(s_{i,j}^\ell, c_{i,j}^\ell, d_{i,j}^\ell)$.

For the PIM architecture with 3D stacked memory, fetching data from DRAM vault costs 2x-10x more time and energy than from on-chip cache in the PE array [14], [15]. In this paper, $P : I, E \mapsto \mathbb{Z}$ associates every intermediate processing result $I_{i,j}$ with two non-negative weights $P_\alpha(I_{i,j})$ and $P_\beta(I_{i,j})$, where $P_\alpha(I_{i,j})$ is the profit to place $I_{i,j}$ in on-chip cache in the PE array, and $P_\beta(I_{i,j})$ is the case for eDRAM in the 3D stacked memory, $P_\alpha(I_{i,j}) \gg P_\beta(I_{i,j})$.

2.3 Re-allocation of Convolution Operations and Motivational Example

Current advanced PIM architecture can only provide 100-300 KB cache capacity for the entire PE array [2]. As a result, the intermediate processing results will consume fairly large storage capacity and postpone the execution of the CNN application. For the sake of illustration, we assume that the PIM architecture consists of four PEs and each data cache of a PE can hold only one intermediate processing result. Based on this configuration, the on-chip cache can concurrently store four intermediate processing results.

Figure 3 illustrates the case that a CNN application runs on four PEs. The graph shown in Figure 2(b) represents this application. Since four PEs can be utilized to process the CNN application, as shown in Figure 3(a), two periods of the application can be concurrently processed. The first period is mapped to PE_0 and PE_1, and the second period is mapped to PE_2 and PE_3, respectively. The following periods of the application adopt the similar pattern. Due to the space constraint of the cache, only $I_{2,4}$ and $I_{2,5}$ in each iteration can be allocated to the cache. Both $I_{3,4}$ and $I_{3,5}$ will be scheduled to the eDRAM. This causes one time unit delay for the execution of convolution tasks T_4 and T_5.

An iteration is the loop body of the schedule that will be iteratively executed. Each iteration contains one or more periods of tasks. It is the hyper-period of the schedule. For the schedule in Figure 3(a), a basic iteration is the same as a period. The schedule length (i.e., execution time) of an iteration is 11 time units. Since two periods of tasks are concurrently executed, the average schedule length for each period takes $11/2 = 5.5$ time units.

In order to use all the computation resources, we adopt the feature of convolution operations and reallocate several convolutions into previous iterations. The newly-added iterations are called *prologue*. The reallocation of convolution operations influences data dependency relations in a given graph, and this technique is normally referred to as retiming. The retiming technique is originally proposed to minimize the cycle period of a synchronous circuit by evenly distributing registers [16]. We extend this technique to map convolution operations to the multiple processing engines in the PIM architecture.

Figure 3(b) illustrates the allocation of convolutional connections and intermediate processing results. From the example, all convolution operations in each iteration are compacted to achieve the minimum execution time. In this schedule, each iteration only takes four time units, and it will be repeatedly executed for every four time units after time unit 12. In this schedule, three iterations (time units 0-12) of retiming operations are allocated into the prologue.

Although the retiming technique can effectively reduce the schedule length in each period by changing the intra-period data dependency into the inter-period data dependency, the generated task schedule still could be improved. The utilization of PEs is not optimized, which will affect the execution time of the entire application. For the schedule in Figure 3(b), four time units are not utilized in each period. The utilization ratio of PEs is 12/16=75%. It is possible to concatenate multiple periods of the application to form one single iteration. This can remove the empty time slots and improve the utilization of computation resources.

To fully exploit parallelism and guarantee the minimum execution time in each iteration for a CNN application, several issues have to be considered. First, intermediate processing result becomes a critical factor in this design and it needs to be carefully allocated. This allocation problem is not trivial, as the execution time of each intermediate processing result $I_{i,j}$ depends on the actual allocation at the cache or the eDRAM. Second, the allocation to the on-chip cache or the eDRAM can directly impact the data dependency of convolution operations, which will further influence both the prologue and the execution time of the CNN application. Third, the capacity constraint of the on-chip cache in the PIM architecture also needs optimization techniques to jointly schedule both convolution operations and intermediate processing results. These observations motive us to propose a novel task-level data allocation framework to exploit parallelism for convolutional connections in the PIM architecture.

3 Para-Net: Exploiting Parallelism for C-NNs with the Minimum Data Movement

3.1 Overview

Neural networks are remarkably adept at various learning systems, but are limited in their ability to store intermediate data over long timescales [17]. In convolutional neural networks, a convolution operation needs to perform the same procedure on one intermediate processing result or another. It merely has to change the address it reads from. Although the emerging PIM architecture provides 3D-stacked layout to address the data movement issue, there still lacks of a data allocation model that can jointly optimize both convolution operations and their intermediate processing results.

Para-Net aims to combine the advantages of parallel processing capability of the PIM architecture and the deterministic property of convolutional connections. In Para-Net, intra-iteration data dependencies among convolution operations are transformed into inter-iteration data dependencies, which allows processing engines to fully utilize their computation resources and significantly boosts the application throughput. In this section, we first analyze the extra data movement for intermediate processing results in Section 3.2. The procedures to generate the initial schedule is presented in Section 3.3. The target problem is further formulated as a dynamic programming model to obtain an optimal solution in Section 3.4.

3.2 The Analysis of Extra Data Movement

In this section, we analyze the bounds of extra data movements for intermediate processing results. For an intermediate processing result $I_{i,j}$ associated with convolutions T_i and T_j, the extra data movement for $I_{i,j}$ is affected by the finish time of T_i and the release time of T_j. We apply the concept of retiming to describe how many iterations of convolution operations are re-allocated into the prologue and how this step affects data dependencies of a given CNN application.

Definition 3.1. *(Retiming) Given a DAG $G = (V, E, P, R)$, retiming R of G is a function that maps each vertex $T_i \in V$ to an integer $R(i)$. $R(i) = 0$ initially; by retiming T_i once, if it is legal, $R(i) = R(i) + 1$, and one iteration of a convolution operation T_i is re-allocated into the prologue.*

For each intermediate processing result $I_{i,j}$, its retiming value $R(i,j)$ denotes how many iterations of intermediate processing result $I_{i,j}$ are re-allocated in the prologue. A retiming function must be legal in order to preserve the semantic correctness. For each pair of convolution operations $(T_i, T_j) \in E$ associated with the intermediate processing result $I_{i,j}$, their retiming functions are legal if $R(i) \geq R(i,j) \geq R(j)$.

Theorem 3.1. *For a pair of convolution operations $(T_i, T_j) \in E$ $(T_i, T_j \in V)$ in the ℓ-th iteration, after retiming T_i for $R(i)$ times and retiming T_j for $R(j)$ times, the associated intermediate processing result $I_{i,j}$ is always schedulable on the PIM architecture if the retimed $T_{i,\ell-R(i)}$ is re-allocated at most two more iterations ahead of the retimed $T_{j,\ell-R(j)}$.*

Proof. By retiming T_i two more iterations ahead of $T_{j,\ell-R(j)}$, T_i will be re-allocated in iteration $\ell - R(j) - 2$. In each iteration, the value of $c_{i,j}$ depends on the allocation to the on-chip cache or the eDRAM, and $c_{i,j} \leq p$. Let $I_{i,j}$ in iteration $\ell - R(j) - 1$ be the associated intermediate processing result of $T_{i,\ell-R(j)-2}$ and $T_{j,\ell-R(j)}$. As $s_i^{\ell-R(j)-2} + c_i^{\ell-R(j)-2} \leq s_{i,j}^{\ell-R(j)-1}$ and $s_{i,j}^{\ell-R(j)-1} + c_{i,j}^{\ell-R(j)-1} \leq s_j^{\ell-R(j)}$, the associated intermediate processing result $I_{i,j}^{\ell-R(j)-1}$ is always schedulable during that time span. □

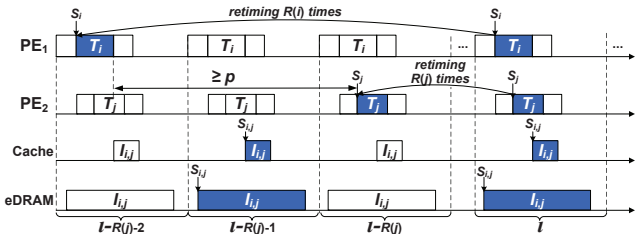


Fig. 4. The exemplary allocation for convolutional connections with data dependency relations.

Theorem 3.1 presents the upper bound of the maximum relative retiming value of each pair of convolutional connections. Based on the definition of retiming value, the data dependency relationship for each pair of convolutional connections will let at most two iterations of T_i into the prologue. Let R_{max} be the maximum retiming value among all convolution operations, $R_{max} = \{\max R(i), T_i \in V\}$. The prologue time can be obtained by $R_{max} \times p$, and this denotes the preprocessing of convolutions before the loop kernel. An example of Theorem 3.1 is given in Figure 4.

For each intermediate processing result $I_{i,j}$, Theorem 3.1 further classifies its allocation to either the on-chip cache or the eDRAM into six cases. Among them, cases 1, 4, and 6 denote that the allocation of $I_{i,j}$ on cache or eDRAM does not affect the relative retiming value. Theses cases will not change the prologue time, so $I_{i,j}$ could be allocated to the eDRAM to save the valuable space in the on-chip cache. For cases 2, 3, and 5, the allocation on the eDRAM will cause at least one extra iteration of prologue time, and these intermediate processing results should compete in the cache capacity.

Figure 6 shows the case with zero relative retiming value. That is, the time span between the finishing time of T_i and the release time of T_j is long enough to hold the intermediate processing result on the eDRAM. The retimed T_i and T_j are in the same period. Therefore, for this case, the relative retiming value $\Delta R(i,j) = R(i) - R(j)$ is equal to zero. The intermediate processing result will not affect the overall retiming value, and it should be assigned to the eDRAM to make room for other intermediate processing

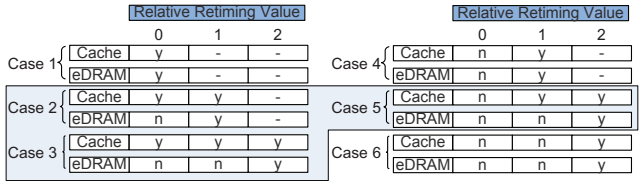


Fig. 5. Six cases for the relative retiming value by allocating an intermediate processing result on either the on-chip cache or the eDRAM.

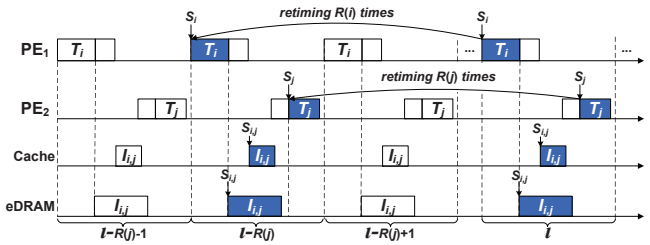


Fig. 6. The case with zero relative retiming value (case-1) by allocating an intermediate processing result on the on-chip cache or the eDRAM.

results that may reduce the prologue time of the schedule. This analysis can further reduce the search space of the dynamic programming model to achieve an efficient solution.

Figure 7 illustrates the case with non-zero relative retiming value for $R(i)$ and $R(j)$, $(T_i, T_j) \in E$. In this example, the time span between the finishing time of T_i and the release time of T_j is adequate to hold the execution of the corresponding intermediate processing result $I_{i,j}$ on the on-chip cache. However, this time span is less than the execution time of $I_{i,j}$ on the eDRAM. If $I_{i,j}$ cannot be allocated on the on-chip cache, T_i has to perform one extra retiming operation to preserve the data dependency.

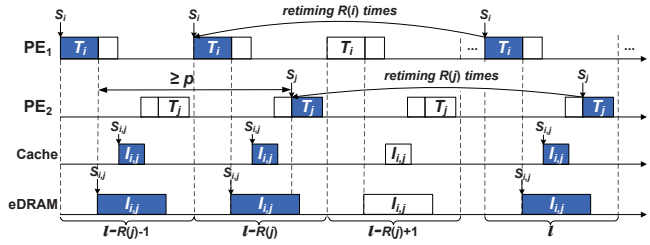


Fig. 7. The case with non-zero relative retiming value (case-2) by allocating an intermediate processing result on the eDRAM.

3.3 Generating the Initial Schedule for Convolutions

Para-Net consists of two major stages to fully exploit the parallelism for CNN applications in the 3D stacked PIM architecture. The first stage involves in the generation of an initial schedule for convolution operations. The initial schedule not only affects the utilization of computation resources, but also determines the allocation of intermediate processing results. Para-Net utilizes a threshold value to define the lower bound of the utilization ratio. The objective is to fully utilize all computation resources provided by the PIM architecture.

Para-Net partitioned the processing engines into N_{grp} groups. Each group will be used to handle one period of convolutions. N_{grp} can be calculated from Equation 1, where N_{max} is the maximum number of convolution or pooling operations that are concurrently processed in one single layer. The value of N_{grp} has two cases. If N_{PE} can be divided with no remainder, the number of PE groups is $\frac{N_{PE}}{N_{max}}$. Otherwise, the value of N_{grp} becomes $\frac{N_{PE}}{N_{max}} + 1$.

$$N_{grp} = \begin{cases} \frac{N_{PE}}{N_{max}}, & \text{if} \quad N_{PE} \% N_{max} = 0, \\ \frac{N_{PE}}{N_{max}} + 1, & \text{if} \quad N_{PE} \% N_{max} \neq 0 \end{cases} \tag{1}$$

Algorithm 3.1 Generate an initial schedule with N_{grp} groups of tasks.

Input: A set of n tasks $\{T_1, T_2, \ldots, T_n\} \in V$, N_{PE} homogeneous PEs, the maximum N_{max} of tasks that are concurrently executed within a convolutional or pooling layer, the threshold PE's utilization ratio U_ζ.
Output: An initial schedule with N_{grp} groups of tasks.
1: Obtain N_{grp} from Equation 1.
2: Initialize N_{grp}.
3: **for** each group GP_i, $i \in [1, N_{grp}]$ **do**
4: **for** each task T_j, $j \in [1, n]$, $T_j \in V$ **do**
5: Assign T_j to a PE $\in GP_i$ with the earliest available time.
6: **end for**
7: **end for**
8: Get the utilization ratio U of PEs from Equation 2.
9: **while** $U < U_\zeta$ **do**
10: **for** each task T_j, $j \in [1, n]$, $T_j \in V$ **do**
11: Assign T_j to a PE in PIM with the earliest available time.
12: **end for**
13: $N_{grp} \leftarrow N_{grp} + 1$.
14: Get the utilization ratio U of PEs from Equation 2.
15: **end while**

The utilization ratio of PEs defines the proportion of the non-idle time for PEs. Its value U can be obtained from Equation 2, where N_{cv} is the total number of convolution or pooling operations, c_i is the execution time of T_i, p is the period of the schedule. Para-Net predefines a threshold utilization ratio U_ζ. The generated schedule should have an utilization ratio that is greater than or equal to U_ζ.

$$U = \frac{\sum_{i=1}^{N_{cv}} c_i \cdot N_{grp}}{p \cdot N_{PE}} \tag{2}$$

Algorithm 3.1 presents the basic steps to generate an initial schedule. Para-Net first divides the PEs into N_{grp} groups based on Equation 1. For each group of PEs, one period of convolution or pooling operations will be scheduled inside the group GP_i, $i \in [1, N_{grp}]$. The PE with the earliest available time will be used to handle the incoming task T_j, $j \in [1, n]$, $T_j \in V$. Note that, for identical groups of PEs, the generated initial schedules are the same. Para-Net will calculate the current utilization ratio U of PEs and compare it with the predefined threshold utilization ratio U_ζ. If the utilization ratio is less than the threshold, Para-Net will assign another period of tasks to improve the utilization ratio. For this set of tasks, all PEs in the PIM architecture could handle the tasks. There is no restriction on which PE can cater the request. The PE with the earliest available time will be selected to process the task.

Figure 8 shows an example to generate the initial schedule of convolutional connections. For the illustration purpose, we assume that there are seven PEs (PE_0 to PE_6) in the PIM architecture, and the threshold utilization ratio is defined as 92%. The modelled CNN application is the same as the one in Figure 2. Based on this configuration, N_{max} is equal to two. and N_{grp} is equal to four. For the first three groups (i.e., $PE_0 \sim PE_1$, $PE_2 \sim PE_3$, and $PE_4 \sim PE_5$), each group consists of two PEs. The last group consists of only one PE (i.e., PE_6). Then the first schedule can be generated, which is shown in Figure 8(a).

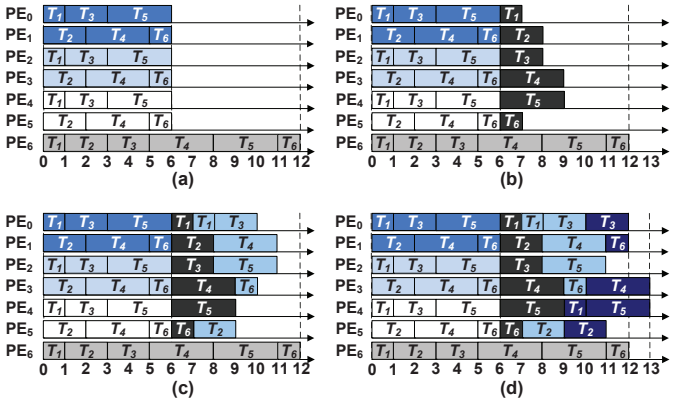


Fig. 8. The exemplary generation of the initial schedule.

At this step, the period of the schedule is 12, and the utilization ratio ($4 \times 12/(7 \times 12) = 57.14\%$) is less than the threshold. Therefore, another period of tasks will be allocated to the empty slots, which is shown in Figure 8(b). Then the utilization ratio is increased to $5 \times 12/(7 \times 12) = 71.43\%$. Since the utilization ratio is still below the threshold, ParaNet will iteratively allocate an extra period of the tasks into the schedule to increase the utilization. The utilization ratio is further improved to $6 \times 12/(7 \times 12) = 85.71\%$ for the schedule in Figure 8(c). Finally, the last period of tasks is allocated, in which the schedule length is also increased to 13. As the utilization ratio becomes $7 \times 12/(7 \times 13) = 92.30\%$, the criteria for the utilization ratio is satisfied. The schedule in Figure 8(d) becomes the initial schedule of the application.

3.4 Optimal Data Allocation for Intermediate Processing Results

Based on the analysis of retiming value, to minimize the prologue time is equivalent to the problem of reducing the maximum retiming value of all convolution operations in the application. This problem has an optimal substructure property. An optimal solution to the problem can be constructed from optimal solutions to subproblems. Therefore, it can be effectively solved by dynamic programming. We now present three steps to construct the optimal data allocation for intermediate processing results.

3.4.1 Characterizing the optimal allocation

The first step of the dynamic programming paradigm is to characterize the structure of an optimal solution. Assume that we know which subset of intermediate processing results can get the maximum profit. The intermediate processing result $I_{i,j}$ with the latest deadline should be scheduled last, such that it will not waste the time between the second last deadline and the last deadline. Therefore, the subset of intermediate processing results that are scheduled will be done in increasing order of deadline.

To give a dynamic programming solution to the problem, we first sort n intermediate processing results. In the sorted order, intermediate processing result I_m will be the m-th task that will be allocated to either the on-chip cache or the eDRAM. This precomputation can be done in time $O(n\log n)$. When we consider to the allocation of an intermediate processing result I_m within the cache capacity S_x, $S_x \in [0, S]$, we can look back at the optimal way to allocate the $m-1$ tasks with the cache capacity S_x. Whether or not to allocate I_m is determined by the current optimal allocation of $m-1$ tasks and the remaining on-chip cache capacity.

3.4.2 A recursive solution

For an intermediate processing result I_m, $1 \le m \le n$, let $\mathbb{B}[S][m]$ be the maximum total profit for a subset of m intermediate processing results $\{I_1, I_2, \ldots, I_m\}$, subject to the cache capacity S. The profit denotes that the maximum reduction on the retiming values with the cache capacity S. Let sp_m be the space requirement to allocate I_m to on-chip cache. Let $\Delta R(m)$ be the reduced retiming value by placing an intermediate processing result I_m on the cache. For example, for case 5 in Figure 5, the retiming values for the on-chip cache and the eDRAM are 1 and 2, respectively. Then $\Delta R(m)$ = 2-1 =1.

The recursive definition of an optimal subproblem is:

$$\mathbb{B}[S][m] = \begin{cases} 0, & \text{if } m = 0, \text{ or } S = 0, \\ 0, & \text{if } m = 1, \text{ and } sp_1 > S, \\ \Delta R(1), & \text{if } m = 1, \text{ and } sp_1 \le S, \\ \mathbb{B}[S][m-1], & \text{if } m > 1, \text{ and } sp_m > S, \\ \max\{\mathbb{B}[S][m-1], & \\ \mathbb{B}[S-sp_m][m-1] + \max\{\Delta R(m) - \Delta R(r_{max}), 0\}\}, & \\ & \text{if } m > 1 \text{ and } sp_m \le S \end{cases} \quad (3)$$

The recursive solution to this problem involves establishing a recurrence for the value of an optimal solution. For the maximum total profit $\mathbb{B}[S][m]$, if either m or the cache capacity S is equal to 0, the profit $\mathbb{B}[S][m]$ becomes 0. This denotes that either no intermediate processing results need for scheduling or the capacity of the cache is equal to zero. If the space requirement to allocate the first intermediate processing result I_1 is larger than the cache capacity S, I_1 cannot be allocated within the on-chip cache. For the case that the space requirement of I_1 is less than or equal to the cache capacity S, the profit is initialized to the reduced retiming value $\Delta R(1)$. If the number of intermediate processing results is greater than one (i.e., $m > 1$) and the cache requirement is larger than the current cache capacity S, this intermediate processing result could not be allocated. This case cannot reduce the retiming value. Then the profit $\mathbb{B}[S][m]$ is initialized to the profit of the previous $m-1$ intermediate processing results (i.e., $\mathbb{B}[S][m]$=$\mathbb{B}[S][m-1]$).

In the recursive formulation, we consider the two subproblems of finding the maximum profit for the set of intermediate processing results $\{I_1, I_2, \ldots, I_n\}$ with different cache capacity. A $n \times S_n$ table $\mathbb{B}$ is needed to generate the maximum total profit $\mathbb{B}[S][m]$, where n is the number of intermediate processing results and d_n is the latest deadline of n intermediate processing results. Since each table entry takes $O(1)$ time to compute, the running time of dynamic programming procedure is $O(n \cdot d_n)$.

3.4.3 Constructing an optimal allocation

The optimal allocation of intermediate processing results can be generated based on the recursive formulation of dynamic programming. Algorithm 3.2 presents the basic

Algorithm 3.2 Generate an optimal allocation from $\mathbb{B}[S][n]$.

Input: A set of n intermediate processing results $\{I_1, I_2, \ldots, I_n\} \subseteq E$, the capacity of the on-chip cache S, and a queue Q.
Output: The allocation and the retiming value of each intermediate processing result I_m, $m \in [1, n]$.
1: **for** each $I_m \in E$ **do**
2: Obtain $\Delta R(m)$.
3: $Enqueue(Q, I_m)$.
4: **end for**
5: Sort I_m in descending order by $\Delta R(m)/sp_m$.
6: **while** $Q \neq \emptyset$ **do**
7: **if** $\mathbb{B}[S - sp_m][m-1] + \max\{\Delta R(m) - \Delta R(r_{max}), 0\}$ is greater than $\mathbb{B}[S][m-1]$ in the dynamic programming model **then**
8: Allocate I_m on the on-chip cache.
9: **else**
10: Allocate I_m on the eDRAM.
11: **end if**
12: $Dequeue(Q, I_m)$.
13: **end while**
14: Return $\mathbb{B}[S][n]$.

procedure to generate the optimal allocation for intermediate processing results. For a set of n intermediate processing results, Para-Net first obtains an initial objective task schedule, which can be obtained from Algorithm 3.1. This initial schedule determines the release time and the finishing time of each computation task, and further determines the status of the corresponding intermediate processing result. Based on this objective schedule and the execution time of each intermediate processing result, $\Delta R(m)$ can be derived from the schedulability test (Line 2).

For intermediate processing results with zero value of $\Delta R(m)$, it denotes that this intermediate processing result will not influence the prologue time. Therefore, these intermediate processing results will be allocated to the eDRAM to make room for other important ones. Then we use a dynamic programming model to select the suitable intermediate processing results and allocate them to the on-chip cache. As a result, not all intermediate processing results will become the inputs of the dynamic programming model.

The intermediate processing results will be inserted into the queue Q (Line 3). This queue defines the allocation order of the intermediate processing results. Para-Net first sorts the tasks by the deadline. For the tasks with the same deadline, Para-Net sorts m intermediate processing results in descending order by the $\Delta R(m)/sp_m$, $m \in [1, n]$.(Line 5). For the intermeidate processing result with the small capacity requirement of the cache and with the large $\Delta R(m)$ will have the priority to be scheduled to the on-chip cache.

Para-Net uses the dynamic programming model to determine the allocation of each intermediate processing result. Specifically, it compares $\mathbb{B}[S - sp_m][m-1] + \max\{\Delta R(m) - \Delta R(r_{max}), 0\}$ with $\mathbb{B}[S][m-1]$ (Line 7). For all the intermeidate processing results that have been allocated to the on-chip cache, each one has a relative retiming value $\Delta R(m)$. $\Delta R(r_{max})$ is the maximum relative retiming value among these values. If $\mathbb{B}[S - sp_m][m-1] + \max\{\Delta R(m) - \Delta R(r_{max}), 0\}$ is greater than $\mathbb{B}[S][m-1]$, this case denotes that I_m can gain more profit. Then I_m should be allocated to the on-chip cache (Line 8). Otherwise, the benefit of I_{m-1} with cache capacity S will have more profit. For this case, the allocation of I_m cannot help increase the profit. The dynamic programming model will iteratively obtain each value in the matrix $\mathbb{B}[S][n]$, and this matrix can generate the optimal allocation for intermediate processing results.

4 Evaluation

To evaluate the effectiveness of the proposed Para-Net, we conduct experiments on various benchmarks from real-life deep learning applications. The objective of the evaluation is to quantify the gains of our approach over the baseline schemes in terms of five performance metrics: throughput, prologue time, utilization ratio of PEs, cache efficiency, and time cost for program execution. In this section, we first introduce the experimental setup and performance metrics. Then we present the experimental results with discussion.

4.1 Experimental Setup

We conduct experiments using several benchmarks from real-life CNN applications. The directed acyclic graphs are abstracted from the widely used deep learning framework Caffe [6]. The neural networks in the training stage is sent to Caffe, and we track the timing and data dependency relations of each convolution or pooling operation. These CNN applications are further partitioned based on the functionality (i.e., convolution, or pooling) to obtain CNN graphs. Among these benchmarks, *CIFAR-10* is an established computer-vision dataset used for object recognition. *AlexNet* and *ZFNet* are task graphs abstracted from training representative neural network models in Caffe. *LeNet* is a test set in the MNIST database of handwritten digits. *image-1* to *image-3* are image processing applications to identify the major objects in the images; *speech* is a voice recognition application to identify the speech of a dedicated speaker.

The evaluation is conducted based on the PIM architecture of Neurocube [8]. The Neurocube neuromorphic architecture is an extension of Micron's Hybrid Memory Cube [18] to support neural computing. It provides a high-density 3D stacked memory to integrate multiple tiers of DRAM and a number of processing engines. In our experiments, the architecture is configured to contain up to 64 processing engines with cross-bar interconnection.

4.2 Experimental Results

1) Throughput: Table 1 presents the throughput of SPARTA [7], Para-CONV [1], and the proposed Para-Net under 16, 32, and 64 PEs. SPARTA is a throughput-aware task allocation approach for many-core platforms. SPARTA collects sensor data to characterize tasks and uses this information to prioritize tasks when performing allocation. Therefore, it is selected for comparison. The basic characteristics of each benchmark are presented in Table 1. Columns "# of vertex" and "# of edge" represent the number of convolution operations, and the number of intermediate processing results, respectively. From the experimental results, our approach can reduce the execution time by 56.48% and 15.67% compared to SPARTA and Para-CONV, respectively.

TABLE 1
The execution time for SPARTA [7], Para-CONV [1] and our Para-Net on 16, 32, and 64 processing engines.

Benchmarks	# of vertex	# of edge	16-PE			32-PE			64-PE		
			SPARTA	Para-CONV	Para-Net	SPARTA	Para-CONV	Para-Net	SPARTA	Para-CONV	Para-Net
CIFAR-10	122	562	31,968	21,861	21,861	37,962	22,407	21,924	56,943	21,217	15,675
AlexNet	106	480	25,974	19,855	19,855	45,954	20,510	19,855	91,908	28,252	19,855
LeNet-1	98	488	55,944	28,434	27,945	62,937	21,252	20,760	104,895	20,080	17,748
LeNet-2	90	285	51,948	28,322	26,806	56,943	21,287	20,660	56,943	15,135	10,410
ZFNet-1	35	139	103,896	72,000	47,329	79,920	49,995	29,203	135,864	67,972	25,225
ZFNet-2	119	576	35,964	26,988	26,988	49,950	28,686	26,988	99,900	32,400	28,269
image-1	71	308	53,946	34,289	32,896	75,924	40,320	33,990	132,867	42,156	29,064
image-2	36	114	59,940	49,980	36,072	39,960	29,988	18,036	95,904	47,972	22,088
image-3	150	847	29,970	20,159	20,159	37,962	20,610	20,197	56,943	18,444	15,825
speech	160	1066	35,964	23,430	23,430	45,954	24,864	23,452	68,931	18,498	18,156

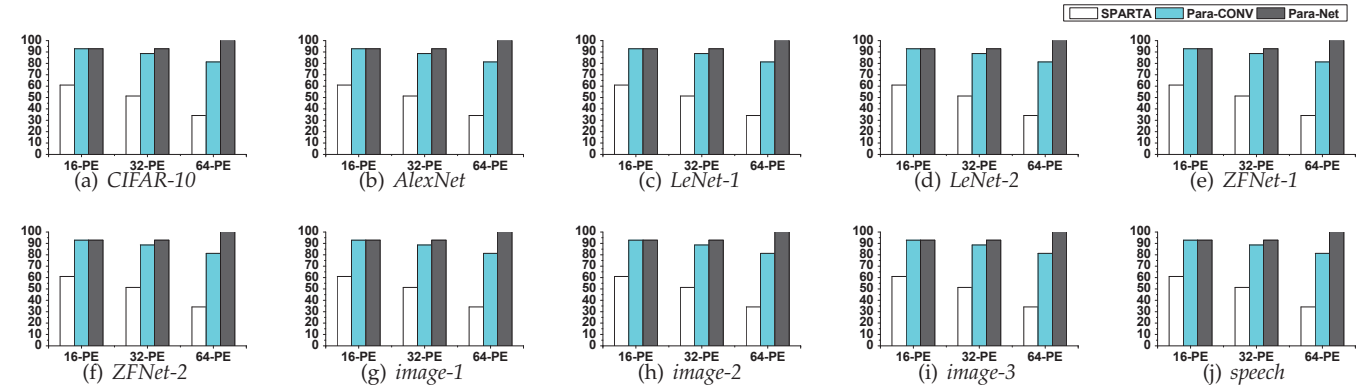


Fig. 9. The utilization ratio of processing engines for SPARTA [7], Para-CONV [1] and our Para-Net on 16, 32, and 64 processing engines.

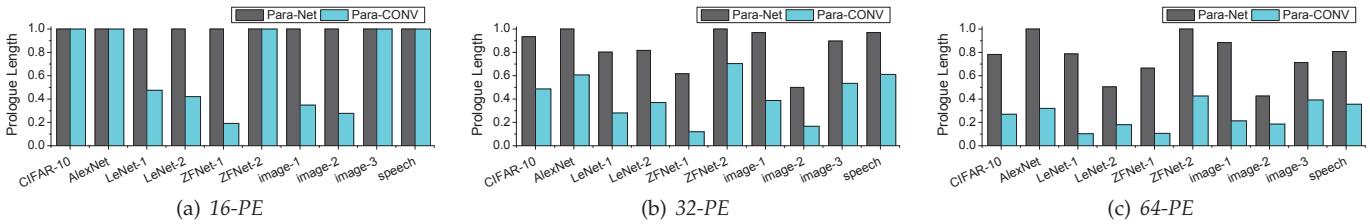


Fig. 10. The prologue time for Para-CONV [1] and our Para-Net on 16, 32, and 64 processing engines.

SPARTA does not adopt retiming operations, and it has to preserve data dependency relationships. Even though the throughput is optimized based on the effective task allocation, the allocation of intermediate processing results does not fully exploit the properties of the PIM architecture. As a result, Para-Net can significantly reduce the execution time of the schedule. Para-CONV is our previous work, which also adopts dynamic programming model to solve the task allocation problem. However, the generation of the initial schedule does not consider the concurrent processing of tasks across different periods. The computation resources are not fully utilized in Para-CONV. The proposed Para-Net further improves this issue to effectively reduce the execution time of the CNN application.

2) Utilization Ratio of PEs: The utilization ratio of processing engines denotes how the CNN applications utilize the computation resources of the PIM architecture. Since the target PIM architecture is memory-bounded, the on-chip cache could be fully utilized. Then the utilization of PEs will affect the throughput and the final task schedule. This performance metric is evaluated, and the results are presented in Figure 9. As expected, Para-Net can achieve the highest utilization ratio for all benchmarks. The utilization ratio of Para-Net ranges between 92.80% and 100.00%, and the average utilization ratio is 95.46%.

Compared to Para-Net, the baseline scheme SPARTA can only use 42.24% of the PEs. We observe that the results for SPARTA are sensitive to the number of PEs. That is, the larger number of PEs leads the lower utilization ratio. These results show that the restrictions on data dependency of SPARTA may prevent it from utilizing all computation resources. Para-CONV aims to fully exploit the parallelism of PEs in the PIM architecture. It will assign tasks to idle PEs to speed up the processing. Therefore, Para-CONV can achieve similar utilization ratios compared to those of Para-Net.

3) Prologue time: Prologue is the preprocessing stage of convolution connections. It will directly impact the total execution time and influence the system throughput. This metric is quantified, and the experimental results are shown in Figure 10. We report the normalized prologue length for Para-CONV and our Para-Net, as they both adopt retiming

TABLE 2
The number of intermediate processing results that are allocated to the on-chip cache for SPARTA [7], Para-CONV [1] and our Para-Net on 16, 32, and 64 processing engines.

Benchmarks	16-PE			32-PE			64-PE		
	SPARTA	Para-CONV	Para-Net	SPARTA	Para-CONV	Para-Net	SPARTA	Para-CONV	Para-Net
CIFAR-10	134	149	149	152	202	302	244	264	510
AlexNet	83	135	135	170	145	271	318	238	542
LeNet-1	96	112	224	109	138	340	161	176	729
LeNet-2	100	135	258	122	149	426	214	192	277
ZFNet-1	28	31	253	30	41	457	62	70	753
ZFNet-2	86	138	138	133	198	270	241	275	790
image-1	83	54	143	98	125	388	164	149	659
image-2	16	30	166	30	36	164	62	66	543
image-3	228	203	203	229	245	411	396	332	598
speech	221	196	196	235	249	349	467	298	725

TABLE 3
The time cost to run SPARTA [7], Para-CONV [1] and our Para-Net on 16, 32, and 64 processing engines.

Benchmarks	16-PE			32-PE			64-PE		
	SPARTA	Para-CONV	Para-Net	SPARTA	Para-CONV	Para-Net	SPARTA	Para-CONV	Para-Net
CIFAR-10	21.14	22.82	28.37	13.74	19.63	28.09	13.77	20.74	32.39
AlexNet	16.02	20.97	22.88	15.21	17.44	25.17	15.15	18.57	35.17
LeNet-1	16.64	18.53	29.47	13.95	16.72	30.92	13.92	15.21	50.21
LeNet-2	15.69	18.23	28.29	12.89	16.06	26.75	12.94	16.62	25.45
ZFNet-1	10.32	13.85	38.40	9.01	13.71	31.37	9.22	14.32	38.28
ZFNet-2	19.89	24.91	29.31	16.04	20.81	30.15	16.32	22.94	48.99
image-1	15.89	17.95	30.76	13.37	17.43	33.35	13.38	19.52	42.76
image-2	10.21	14.54	30.00	9.05	13.59	22.77	9.13	13.61	33.10
image-3	22.23	23.77	26.37	17.10	19.83	31.53	16.81	30.49	34.96
speech	25.25	25.12	28.35	19.08	23.12	35.07	19.62	34.10	49.67

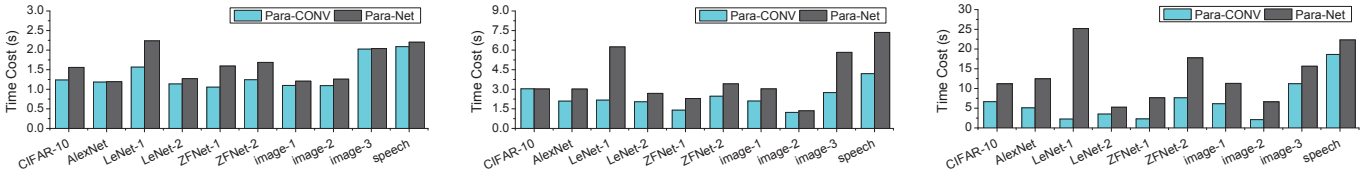
operations. The experimental results are normalized by the prologue of Para-Net on 16 PEs.

Para-CONV gets the same prologue length as Para-Net for five benchmarks (*CIFAR-10*, *AlexNet*, *ZFNet-2*, *image-3*, and *speech*) running on 16 PEs. With the increasing of the number of PEs, the prologue length correspondingly decreases, and Para-Net experiences a longer prologue length compared to Para-CONV. This is mainly due to the generation of the initial schedule for convolutions. Para-CONV transforms directed acyclic graphs as independent task nodes. The execution of tasks in one period forms one iteration. Our Para-Net integrates multiple periods of tasks into one iteration in order to guarantee the utilization ratio of PEs. Therefore, Para-Net may introduce multiple periods of prologue. Note that, the prologue is only executed for once. Compared to the benefit gained from the significant reduction of execution time in each iteration, the execution of prologue will not cause significant timing overhead.

4) Cache Efficiency: Para-Net aims to allocate intermediate processing results to appropriate location (on-chip cache or eDRAM) to improve the system throughput. Table 2 illustrates the number of intermediate processing results that are allocated to the on-chip cache on 16, 32, and 64 processing engines. From the results, more intermediate processing results are allocated to the on-chip cache when the configuration changes from 16 PEs to 32 PEs or 64 PEs. For the results with 16 PEs, the cache efficiency for SPARTA is similar to Para-CONV for most benchmarks. For benchmarks *image-3* and *speech*, SPARTA allocates the largest number of intermediate processing results on the on-chip cache. This is because, SPARTA does not change the data dependency relationship, and it does not allocate multiple periods of CNN applications on the same PE to concurrently utilize its computation capability. The on-chip cache is not shared by multiple iterations, which leads to the efficiency of the cache allocation. With the increase of the number of PEs, SPARTA suffers from a low PE utilization ratio. Then the cache efficiency for SPARTA is relatively worse than that for Para-Net.

We also observe that, the number of intermediate processing results that are allocated to the on-chip cache may not directly affect the schedule length and the prologue length, which are presented in Table 1 and Figure 10, respectively. The target PIM architecture is memory-bounded, so the utilization ratio of the on-chip cache should be close to 100%. Since both Para-CONV and Para-Net adopt dynamic programming models to optimally allocate the allocation of intermediate processing results, they could select the most suitable candidate to fully utilize the resource-constrained on-chip cache. Compared to Para-CONV, the proposed Para-Net further improves the actual number of intermediate processing results that are allocated to the on-chip cache. The experimental results show that, Para-Net can allocate 2.25x intermediate processing results compared to that of Para-CONV.

5) Time Cost for Program Execution: The time cost to run each schedule algorithm represents its time complexity. This performance metric is quantified, and the results are shown in Table 3. As Para-CONV and Para-Net adopt dynamic programming models to solve the target problem, they should experience longer processing time compared to the heuristic-based algorithms. However, from the results, the



(a) *16-PE* (b) *32-PE* (c) *64-PE*

Fig. 11. The time cost for dynamic programming of Para-CONV [1] and our Para-Net on 16, 32, and 64 processing engines.

time cost for three scheduling algorithms are at the same scale. That is because, the dynamic programming models in Para-CONV and Para-Net are based on the initial schedule. Not all intermediate processing results will be selected as the inputs of the dynamic programming model. The schedulability analysis also limits the upper bound for retiming operations. As a result, the time cost for the proposed Para-Net does not incur significant overhead as other conventional dynamic programming based algorithms.

We also compare the time cost for the execution of the dynamic programming model for Para-CONV and the proposed Para-Net on 16, 32, and 64 processing engines. The experimental results are shown in Figure 11. We found that Para-Net takes a little bit longer time for the execution of the dynamic programming model compared to that of Para-CONV. Para-Net maintains multiple periods of the applications to form one iteration. Then the dynamic programming model needs to handle a longer period of time. Note that the preprocessing of the dynamic programming model belongs to static scheduling. It can be done at the compile stage, and it will not affect the execution of the actual CNN applications.

5 Related Work

Hardware Accelerators for CNNs. Hardware accelerators become viable solutions to speed up the processing of CNN applications. Different from conventional general purpose graphics processing unit (GPGPU), hardware accelerators adopt FPGA (Field-Programmable Gate Array) or ASIC (Application Specific Integrated Circuit) to cater the specific characteristics of CNN applications. There have been several FPGA implementations [19], [20], [21], [22] to provide computationally and power efficient systems [23]. However, once a neural network engine is synthesized, FPGA platforms cannot be programmed on-line [8]. There have been a number of ASIC designs [24], [25], which can obtain even better performance compared to those of FPGA platforms. The major limitation of the ASIC design is its scalability of on-chip memory or system interface with off-chip memory. Para-Net adopts a different strategy to accelerate the processing of CNN applications. Para-Net is a software-based scheduling framework. Therefore, it can provide a flexible adoption for any CNN application on any hardware platform. Para-Net can be combined with FPGA or ASIC platforms to provide the scalability of parallelism and to further improve the computational efficiency.

System Software and Interface for PIM. There have been many software-based solutions to improve the throughput or reduce the access latency for the emerging PIM architecture. Some techniques aim to place computational resources inside or near storage to form in-storage computing or near-data processing [26], [27]. Some other techniques consider system software support for the PIM architecture with non-volatile memory [28], [29], [30]. In such systems, non-volatile memory could be attached to memory bus to form the hybrid memory architecture and reduce the I/O latency. The hardware implementation of such PIM architecture could be flash-based memory-channel NVDIMM like eXFlash [31].

Para-Net focuses on the 3D-stacked PIM architecture like Hybrid Memory Cubes [18]. Other emerging memory technologies like metal-oxide resistive random-access memory (RRAM) [32], Memristor [33], or other types of persistent memory [34] could also be the typical hardware implementation of the PIM architecture. Para-Net is a system software technique, which aims to improve the efficiency of cache and to reduce the data movement from memory. Para-Net can be built on the above mentioned PIM architectures. Both system abstractions of applications and the modeling of memory resources can be applied to other PIM architectures to further improve the efficiency of these PIM architectures.

Task Scheduling for Dataflow Applications. Data-intensive applications such as streaming applications are commonly modelled as directed acyclic graphs or synchronous data flow graphs [35], [36]. Task scheduling for streaming applications can effectively exploit the parallelism of the system architecture. There have been many scheduling techniques for streaming applications on multiprocessor or multicore systems to optimize response time, reduce energy consumption, and enhance reliability [37], [38], [39], [40]. Some other scheduling techniques aim to jointly optimize both computation and communication resources [41], [42], [43], [44], [45]. Although the target applications of this paper (i.e., CNN applications) can also be modelled as directed acyclic graphs, they have several special characteristics. The graph abstraction of CNN applications is different from general directed acyclic graphs. Therefore, the scheduling techniques for streaming applications cannot be directly applied to solve the task mapping and task scheduling problem in CNNs.

Data Allocation and Management on PIM. Some previous work studied the data allocation and management for data-intensive workloads on the GPU architecture [46] and other emerging PIM architectures [47], [48], [49]. They can provide either the computation bounded design or the memory bounded design to speed up the processing of workloads on PIM. Chen et al. proposed a CNN accelerator called Eyeriss [2], to exploit zero valued neurons and to reduce power consumption. Our Para-Net is different from

the above techniques. Para-Net aims to find a balance point for computation and memory resources. It is a general scheduling technique that can be applied to different PIM architectures to exploit the computational capability and hide the extra latency from memory. Therefore, Para-Net can be combined with the above mentioned PIM-based data allocation techniques to further improve the efficiency of the schedule.

6 Conclusion

This paper presents a data allocation and scheduling framework, called Para-Net, to minimize data movement for convolutional connections in the PIM architecture. Para-Net exploits the parallelism for convolutions to generate the initial schedule. The target problem is transformed as the task scheduling for intermediate processing results based on the initial schedule. We perform schedulability analysis and formulate the problem as a dynamic programming model. The evaluation is conducted based on the widely used deep learning framework Caffe. The experimental results show that Para-Net can effectively reduce data movement and improve the processing time, while incurring negligible time for preprocessing.

In the future, we plan to investigate the use of our approach on other emerging PIM architectures and propose a general model that can be adaptively applied to different system architectures. We also plan to study energy issue for the PIM architecture with CNN applications, which would provide potential opportunities of the PIM-customized design.

References

[1] Y. Wang, M. Zhang, and J. Yang, "Exploiting parallelism for convolutional connections in processing-in-memory architecture," in *2017 54th ACM/EDAC/IEEE Design Automation Conference (DAC)*, June 2017, pp. 1–6.

[2] Y. H. Chen, J. Emer, and V. Sze, "Eyeriss: A spatial architecture for energy-efficient dataflow for convolutional neural networks," in *2016 ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA)*, June 2016, pp. 367–379.

[3] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," in *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2015, pp. 1–9.

[4] J. Albericio, P. Judd, T. Hetherington, T. Aamodt, N. E. Jerger, and A. Moshovos, "Cnvlutin: Ineffectual-neuron-free deep neural network computing," in *2016 ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA)*, June 2016, pp. 1–13.

[5] S. Li, C. Xu, Q. Zou, J. Zhao, Y. Lu, and Y. Xie, "Pinatubo: A processing-in-memory architecture for bulk bitwise operations in emerging non-volatile memories," in *2016 53nd ACM/EDAC/IEEE Design Automation Conference (DAC)*, June 2016, pp. 1–6.

[6] 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.

[7] 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.

[8] 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.

[9] J. Lee, J. H. Ahn, and K. Choi, "Buffered compares: Excavating the hidden parallelism inside DRAM architectures with lightweight logic," in *2016 Design, Automation Test in Europe Conference Exhibition (DATE)*, March 2016, pp. 1243–1248.

[10] J. Chung and T. Shin, "Simplifying deep neural networks for neuromorphic architectures," in *2016 53nd ACM/EDAC/IEEE Design Automation Conference (DAC)*, June 2016, pp. 1–6.

[11] 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.

[12] L. Song, Y. Wang, Y. Han, X. Zhao, B. Liu, and X. Li, "C-brain: A deep learning accelerator that tames the diversity of CNNs through adaptive data-level parallelization," in *2016 53nd ACM/EDAC/IEEE Design Automation Conference (DAC)*, June 2016, pp. 1–6.

[13] W. Wen, C. Wu, Y. Wang, K. Nixon, Q. Wu, M. Barnell, H. Li, and Y. Chen, "A new learning method for inference accuracy, core occupation, and performance co-optimization on TrueNorth chip," in *2016 53nd ACM/EDAC/IEEE Design Automation Conference (DAC)*, 2016, pp. 1–6.

[14] M. Horowitz, "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)*, Feb 2014, pp. 10–14.

[15] M. Poremba, S. Mittal, D. Li, J. S. Vetter, and Y. Xie, "Destiny: A tool for modeling emerging 3D NVM and eDRAM caches," in *2015 Design, Automation Test in Europe Conference Exhibition (DATE)*, March 2015, pp. 1543–1546.

[16] N. L. Passos and E. H. M. Sha, "Synchronous circuit optimization via multidimensional retiming," *IEEE Transactions on Circuits and Systems II: Analog and Digital Signal Processing*, vol. 43, no. 7, pp. 507–519, Jul 1996.

[17] A. Graves et al., "Hybrid computing using a neural network with dynamic external memory," *Nature*, vol. 538, no. 7626, pp. 1–6, Oct. 2016.

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

[19] 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.

[20] Y. Wang, J. Xu, Y. Han, H. Li, and X. Li, "DeepBurning: Automatic generation of FPGA-based learning accelerators for the neural network family," in *2016 53nd ACM/EDAC/IEEE Design Automation Conference (DAC)*, June 2016, pp. 1–6.

[21] M. Alwani, H. Chen, M. Ferdman, and P. Milder, "Fused-layer CNN accelerators," in *2016 49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, Oct 2016, pp. 1–12.

[22] 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, 2018.

[23] A. Rahman, J. Lee, and K. Choi, "Efficient FPGA acceleration of convolutional neural networks using logical-3D compute array," in *2016 Design, Automation Test in Europe Conference Exhibition (DATE)*, March 2016, pp. 1393–1398.

[24] 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.

[25] R. Andri, L. Cavigelli, D. Rossi, and L. Benini, "YodaNN: An architecture for ultra-low power binary-weight CNN acceleration," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 37, no. 1, pp. 48–60, 2018.

[26] J. Wang, D. Park, Y. Papakonstantinou, and S. Swanson, "SSD in-storage computing for search engines," *IEEE Transactions on Computers*, vol. PP, no. 99, pp. 1–1, 2016.

[27] W. Wen, J. Yang, and Y. Zhang, "Optimizing power efficiency for 3D stacked GPU-in-memory architecture," *Microprocessors and Microsystems*, vol. 49, pp. 44 – 53, 2017.

[28] D. Liu, K. Zhong, X. Zhu, Y. Li, L. Long, and Z. Shao, "Non-volatile memory based page swapping for building high-performance mobile devices," *IEEE Transactions on Computers*, vol. 66, no. 11, pp. 1918–1931, 2017.

[29] F. R. Poursafaei, M. Bazzaz, and A. Ejlali, "NPAM: NVM-aware page allocation for multi-core embedded systems," *IEEE Transactions on Computers*, vol. 66, no. 10, pp. 1703–1716, 2017.

[30] J. Choi and G. H. Park, "NVM way allocation scheme to reduce NVM writes for hybrid cache architecture in chip-multiprocessors,"

IEEE Transactions on Parallel and Distributed Systems, vol. 28, no. 10, pp. 2896–2910, 2017.
[31] Lenovo Group Ltd., "http://www.lenovo.com/images/products/system-x/pdfs/datasheets/ exflash_memory_channel_storage.pdf ," 2018.
[32] P. Chi, S. Li, C. Xu, T. Zhang, J. Zhao, Y. Liu, Y. Wang, and Y. Xie, "PRIME: A novel processing-in-memory architecture for neural network computation in ReRAM-based main memory," in *2016 ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA)*, June 2016, pp. 27–39.
[33] L. Xia, B. Li, T. Tang, P. Gu, P. Y. Chen, S. Yu, Y. Cao, Y. Wang, Y. Xie, and H. Yang, "MNSIM: Simulation platform for memristor-based neuromorphic computing system," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. PP, no. 99, pp. 1–1, 2017.
[34] Y. Lu, J. Shu, L. Sun, and O. Mutlu, "Improving performance and endurance of persistent memory with loose-ordering consistency," *IEEE Transactions on Parallel and Distributed Systems*, vol. PP, no. 99, pp. 1–1, 2017.
[35] 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.
[36] Y. Wang, Z. Shao, H. C. B. Chan, D. Liu, and Y. Guan, "Memory-aware task scheduling with communication overhead minimization for streaming applications on bus-based multiprocessor system-on-chips," *IEEE Transactions on Parallel and Distributed Systems*, vol. 25, no. 7, pp. 1797–1807, July 2014.
[37] J. J. Han, X. Tao, D. Zhu, H. Aydin, Z. Shao, and L. T. Yang, "Multicore mixed-criticality systems: Partitioned scheduling and utilization bound," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 37, no. 1, pp. 21–34, 2018.
[38] K. Chronaki, A. Rico, M. Casas, M. Moret, R. M. Badia, E. Ayguad, J. Labarta, and M. Valero, "Task scheduling techniques for asymmetric multi-core systems," *IEEE Transactions on Parallel and Distributed Systems*, vol. 28, no. 7, pp. 2074–2087, July 2017.
[39] P.-J. Micolet, A. Smith, and C. Dubach, "A machine learning approach to mapping streaming workloads to dynamic multicore processors," in *Proceedings of the 17th ACM SIGPLAN/SIGBED Conference on Languages, Compilers, Tools, and Theory for Embedded Systems (LCTES)*, 2016, pp. 113–122.
[40] Z. Wang, J. Yang, R. Melhem, B. Childers, Y. Zhang, and M. Guo, "Simultaneous multikernel GPU: Multi-tasking throughput processors via fine-grained sharing," in *2016 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, March 2016, pp. 358–369.
[41] A. Minaeva, B. Akesson, Z. Hanzalek, and D. Dasari, "Time-triggered co-scheduling of computation and communication with jitter requirements," *IEEE Transactions on Computers*, vol. 67, no. 1, pp. 115–129, 2018.
[42] H. Yun, W. Ali, S. Gondi, and S. Biswas, "BWLOCK: A dynamic memory access control framework for soft real-time applications on multicore platforms," *IEEE Transactions on Computers*, vol. 66, no. 7, pp. 1247–1252, July 2017.
[43] 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.
[44] 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.
[45] X. Zhu, J. Wu, and T. Li, "Leveraging time prediction and error compensation to enhance the scalability of parallel multi-core simulations," *IEEE Transactions on Parallel and Distributed Systems*, vol. 28, no. 9, pp. 2553–2566, Sept 2017.
[46] Z. Wang, J. Yang, R. Melhem, B. Childers, Y. Zhang, and M. Guo, "Quality of service support for fine-grained sharing on GPUs," in *2017 ACM/IEEE 44th Annual International Symposium on Computer Architecture (ISCA)*, June 2017, pp. 269–281.
[47] B. Akin, F. Franchetti, and J. C. Hoe, "Data reorganization in memory using 3D-stacked DRAM," in *2015 ACM/IEEE 42nd Annual International Symposium on Computer Architecture (ISCA)*, June 2015, pp. 131–143.
[48] M. Gao, G. Ayers, and C. Kozyrakis, "Practical near-data processing for in-memory analytics frameworks," in *2015 International Conference on Parallel Architecture and Compilation (PACT)*, Oct 2015, pp. 113–124.
[49] M. Gao and C. Kozyrakis, "HRL: Efficient and flexible reconfigurable logic for near-data processing," in *2016 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, March 2016, pp. 126–137.

Yi Wang received the Ph.D. degree in computer science from the Department of Computing, the Hong Kong Polytechnic University in 2011. He received his B.E. and M.E. degrees in electrical engineering from Harbin Institute of Technology, Harbin, China, in 2005 and 2008, respectively. He is currently an associate professor with the College of Computer Science and Software Engineering, Shenzhen University, Shenzhen, China. His research interests include embedded systems, non-volatile memory, and real-time scheduling for multi-core systems. Dr. Wang won the best paper award in LCTES 2017 and the best paper award nominee in ASP-DAC 2015.

Weixuan Chen is currently a Ph.D. student in the College of Computer Science and Software Engineering, Shenzhen University, Shenzhen, China. He received the B.E. degree in Computer Science from Shenzhen University, in 2017. His research interests include system-level design and implementation, real-time systems, and emerging memory techniques for embedded systems.

Jing Yang received her B.E. and M.E. degrees in electrical engineering from Harbin Institute of Technology, Harbin, China, in 2006 and 2008, respectively. She is currently an assistant professor at Experimental and Innovation Practice Center, Harbin Institute of Technology, Shenzhen. Before joining Harbin Institute of Technology, she was a senior engineer at State Grid Electric Power Research Institute, from 2008 to 2016. Her research interests include cyber-physical systems, power electronics and drive control techniques, and artificial intelligence in industrial systems.

Tao Li received graduate degree in computer engineering from Northwestern Polytechnical University, China, in 1993 and the PhD degree in computer engineering from the University of Texas at Austin, in 2004. He is currently a full professor in the Department of Electrical and Computer Engineering, University of Florida. His research interests include computer architecture, microprocessor, memory and storage system design, virtualization technologies, energy-efficient, sustainable/ dependable data center, cloud/big data computing platforms, the impacts of emerging technologies and applications on computing, and evaluation of computer systems. He has published more than 90 refereed papers in different journals and conferences in these fields. Dr. Tao Li co-authored two papers that won the Best Paper Awards in ICCD 2016, HPCA 2011 and six papers that were nominated for the Best Paper Awards in HPCA 2017, ICPP 2015, CGO 2014, DSN 2011, MICRO 2008 and MASCOTS 2006. He is a fellow of the IEEE.