Exploiting Parallelism for Convolutional Connections in Processing-In-Memory Architecture

Yi Wang[1], Mingxu Zhang[1,3], Jing Yang[2]
[1] College of Computer Science and Software Engineering, Shenzhen University
[2] Experimental and Innovation Practice Center, Harbin Institute of Technology, Shenzhen
[3]State Key Laboratory of Computer Architecture, Institute of Computing Technology, Chinese Academy of Sciences
yiwang@szu.edu.cn

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 recent development in processing-in-memory (PIM) architecture seeks to minimize data movement by computing the data at the dedicated non-volatile device, how to jointly explore the computation capability of PIM and utilize the highly parallel property of neural network remains a critical issue.

This paper presents *Para-CONV*, that exploits parallelism for deterministic convolutional connections in PIM architecture. Para-CONV achieves data-level parallelism for convolutions by fully utilizing the on-chip processing engine (PE) in PIM. The objective is to minimize data movement and data fetching from off-PE DRAM for inter-PE communications. We formulate this data allocation problem as a dynamic programming model and obtain an optimal solution. Para-CONV is evaluated through a set of benchmarks from both real-life CNN applications and synthetic task graphs. The experimental results show that Para-CONV can significantly improve the throughput and reduce data movement compared with the baseline scheme.

CCS Concepts

•**Hardware** → Memory and dense storage; Neural systems; Non-volatile memory; Communication hardware, interfaces and storage; •**Computer systems organization** → Embedded systems; •**Software and its engineering** → Allocation/deallocation strategies; Scheduling;

Keywords

Near-data processing; neuromorphic computing; non-volatile memory; scheduling; parallel computing

1. INTRODUCTION

Computational efficiency is a primary concern in deep learning systems, where hundreds of filters and channels in the high-dimensional convolutions have to be simultaneously processed. Current state-of-the-art convolutional neural networks (CNNs) require several hundreds of megabytes for filter weight storage and 30K-600K operations per input pixel [3, 16]. In such networks, convolutions normally take up about 90% of the CNN operations, and they dominate execution time [2]. The expensive data movement for convolutions poses throughput challenges to the underlying computing and storage hardware.

Processing-in-memory (PIM) or the emerging near-data processing (NDP) is a promising solution to address the data movement challenges [12]. 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. 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-CONV*, a novel task-level data allocation framework to exploit parallelism for convolutional connections in PIM architecture. Para-CONV 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-CONV captures the highly parallel feature of convolutions and lets a number of convolutions reschedule into earlier iterations. After transforming 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.

To minimize 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. To the best of our knowledge, this is the first work that jointly opti-

DAC '17, June 18-22, 2017, Austin, TX, USA
© 2017 ACM. ISBN 978-1-4503-4927-7/17/06…$15.00
DOI: http://dx.doi.org/10.1145/3061639.3062242

mizes convolutions and intermediate data in PIM architecture.

We evaluate the proposed technique with a set of benchmarks from both real-life convolutional applications and synthetic task graphs. Several task graphs are obtained from GoogLeNet ConvNet [16]. Synthetic graphs with over 500 convolutions are also used in the experiments. We compare Para-CONV with the baseline scheme [6] in terms of throughput, and evaluate the scalability and off-chip fetching penalty. Experimental results show that, Para-CONV can achieve an average 53.42% reduction in execution time and effectively utilize the PIM architecture with the minimum overall data movement penalty.

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 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 [11]. 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 TSVs. Several novel structures also adopt emerging metal-oxide resistive random-access memory (RRAM) or neuromorphic circuit to process neural networks [4, 9, 18].

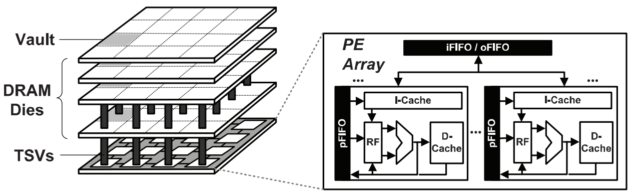

Figure 1: System architecture of accelerator system for CNN processing.

2.2 Application Model for CNN

Convolutional neural network is designed to recognize visual patterns directly from pixel images with minimal preprocessing [5]. 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* [19]. The primary computation of CNN is in the convolutional layers. Each convolutional layer applies several three-dimensional filters over a three dimensional input [2]. 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.

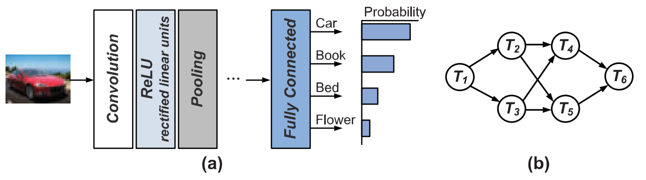

Figure 2: (a) The standard procedure for CNN with multiple stacked layers. (b) A CNN can be modelled as a directed acyclic graph.

A CNN can be represented by a directed acyclic graph [3, 10]. 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, $(V_i, V_j) \in E$ $(V_i, V_j \in V)$, represents the intermediate processing results generated from vertex V_i and requested by vertex V_j. For each directed edge $(V_i, V_j) \in E$, an intermediate processing result $I_{i,j}$ denotes the corresponding data transfer from convolution operation V_i to convolution operation V_j.

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 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 [15, 17]. Therefore, a CNN can be further modelled as a periodically executed dataflow with deterministic convolutional connections.

Let p be the iteration (or period) of each convolution operation V_i and that of each associated intermediate processing result $I_{i,j}$. Convolution operation V_i is associated with three tuples $V_i(s_i, c_i, d_i)$, where s_i is the start time, c_i is the execution time, and d_i is the deadline of V_i. For V_i in the ℓ-th iteration, V_i^ℓ, three tuples become $V_i^\ell(s_i^\ell, c_i^\ell, d_i^\ell)$, 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 [7, 14]. 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 100-300KB cache capacity for the entire PE array [3]. 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 CNN application, as shown in Figure 3(a), two iterations of the application can be concurrently processed. The first iteration is mapped to PE1 and PE2, and the second iteration is mapped to PE3 and PE4. Due to the space constraint of cache, only $I_{2,4}$ and $I_{2,5}$ in each iteration can be allocated to cache. Both $I_{3,4}$ and $I_{3,5}$ will be scheduled to eDRAM.

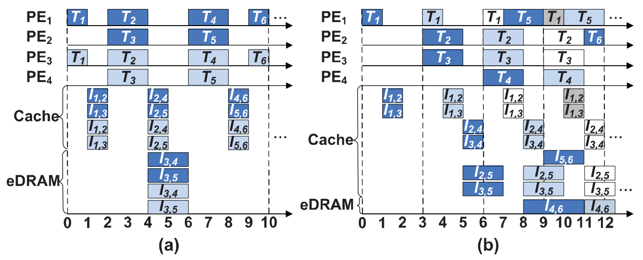

Figure 3: (a) Intermediate processing results cause delay of convolution execution. (b) Fully exploiting parallelism for convolution execution needs joint optimization of convolution operations and intermediate processing results.

This causes one time unit delay for the execution of convolution tasks T_4 and T_5.

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 [13]. We extend this technique to map convolution operations to the multiple processing engines in 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. Each iteration takes only three time units, and it will be repeatedly executed for every three time units after time unit 9. In this schedule, three iterations of retiming (time units 0-9) are allocated into prologue.

To fully exploit parallelism and guarantee the minimum execution time in each iteration for a CNN application, several issues have to be considered. Intermediate processing results become the critical factor in this design and need 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 cache or eDRAM. The allocation to cache or eDRAM can directly impact the data dependency of convolution operations, which will further influence both prologue and execution time of the CNN application. The capacity constraint of 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 PIM architecture.

3. PARA-CONV: PARALLELISM FOR CONVOLUTIONAL CONNECTIONS WITH 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 [1]. 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-CONV aims to combine the advantages of parallel processing capability of PIM and the deterministic property of convolutional connections. In Para-CONV, 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 target problem is further formulated as a dynamic programming model to obtain an optimal solution in Section 3.3.

3.2 The Analysis of Extra Data Movement

In this section, we analyze the bounds of extra data movement for intermediate processing result. For an intermediate processing result $I_{i,j}$ associated with convolutions V_i and V_j, the extra data movement for $I_{i,j}$ is affected by the finish time of V_i and the start time of V_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 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 intermediate processing result $I_{i,j}$, their retiming function is 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 PIM 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 $\ell - R(T_j) - 2$. In each iteration, the value of $c_{i,j}$ depends on the allocation to on-chip cache or eDRAM, and $c_{i,j} \leq p$. Let $I_{i,j}$ in iteration $\ell - R(T_j) - 1$ 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. □

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 relation 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, $R_{max} = \{\max R(T_i), T_i \in V\}$. The prologue time can be obtained by $R_{max} \times p$, and this denotes the preprocessing of convolutions before the loop kernel.

For each intermediate processing result $I_{i,j}$, Theorem 3.1 further classifies its allocation to either on-chip cache or eDRAM into six

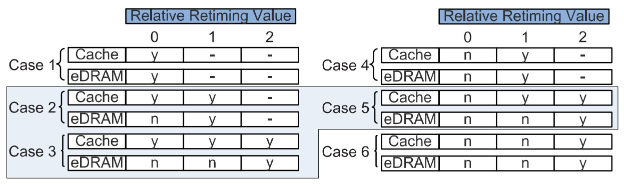

Figure 4: Six cases for the relative retiming value by allocating intermediate processing result on either on-chip cache or eDRAM.

cases. Among them, cases 1, 4, and 6 denotes 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 eDRAM to save the valuable space in on-chip cache. For cases 2, 3, and 5, the allocation on eDRAM will cause at least one extra iteration of prologue time, and these intermediate processing results should compete in the cache capacity.

3.3 Optimal Data Allocation for Convolutional Connections

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 convolutional connections.

3.3.1 *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 according to deadline $d_{i,j}$. In the sorted order, intermediate processing result I_m has the m-th deadline, $m \in [1, n]$. This precomputation can be done in time $O(n\log n)$. When we consider to allocate intermediate processing result I_m within the deadline d_m, we can look back at the optimal way to allocate the $m-1$ tasks within the deadline d_{m-1} and whether or not to allocate I_m to the on-chip cache.

3.3.2 *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 4, the retiming values for on-chip cache and eDRAM are 1 and 2, respectively. Then $\Delta R(m)$ = 2-1 =1.

The recursive definition of an optimal subproblem is:

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

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 $\mathbb{B}[S, m]$ is 0. If the space requirement to allocate the first intermediate processing result I_1 is larger than 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 cache capacity S, the profit is initialized to the reduced retiming value $\Delta R(1)$.

In the recursive formulation, we consider the two subproblems of finding the maximum profit for the set of intermediate processing results $\{I_1, I_2, \ldots, I_m\}$ 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 $O(n \cdot d_n)$.

3.3.3 *Constructing an optimal allocation*

The optimal allocation of intermediate processing results can be generated based on the recursive formulation of dynamic programming. For a set of n intermediate processing results, Para-CONV first obtain an initial objective task schedule, which is known-priori. Based on this objective schedule and the execution time of each intermediate processing result, $\Delta R(m)$ can be derived. For intermediate processing results with zero value of $\Delta R(m)$, this denotes that this intermediate processing result will not influence the prologue time. Therefore, these intermediate processing results will be allocated to on-chip cache to make room for other important ones. Then we use dynamic programming model to select the suitable intermediate processing results and allocate them to on-chip cache.

4. EVALUATION

To evaluate the effectiveness of the proposed Para-CONV, we conduct experiments on various benchmarks from both real-life deep learning applications and synthetic task graphs. The objective is of the evaluation is to quantify the gains of our approach over the baseline schemes in terms of three performance metrics: throughput, prologue time and cache utilization. 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 perform simulations on various benchmarks from both real-life CNN applications and synthetic task graphs. Several real-life CNN applications are obtained from benchmark GoogLeNet ConvNet [16]. The synthetic task graphs were generated by running several CNN applications. Among them, *cat*, *car*, and *flower* are image processing applications that identify the object in the image. *Character* is character recognition application that recognize the bitmap pattern of handwritten characters. *Image-compress* processes vast amounts of information to make image compression. *Stock-predict*, *string-matching*, and *shortest-path*, *speech* are applications to perform stock prediction, match the same strings, find shortest path, and recognize speech of a dedicated speaker. *Protein* is an application that can analyze the content of protein.

Table 1: **Total execution time of SPARTA [6] and our Para-CONV on 16, 32, and 64 processing elements.**

Benchmarks	# of vertex	# of edge	16-core			32-core			64-core		
			SPARTA	Para-CONV	IMP (%)	SPARTA	Para-CONV	IMP (%)	SPARTA	Para-CONV	IMP (%)
cat	9	21	4.7	4.0	85.13	3.3	1.5	46.35	1.2	0.6	51.06
car	13	28	15.0	5.4	36.02	7.5	3.3	44.00	3.8	0.6	16.00
flower	21	51	18.7	9.9	52.97	9.4	4.5	48.16	4.7	3.3	70.63
character-1	46	121	35.1	17.7	50.48	17.6	8.7	49.63	8.8	3.6	41.08
character-2	52	130	45.2	22.2	49.18	22.6	12.3	54.50	11.3	6.3	55.84
image-compress	70	178	56.9	27.0	47.54	28.5	13.2	46.50	14.2	5.1	35.96
stock-predict	83	218	64.5	31.6	48.94	32.3	18.0	55.95	16.1	7.5	46.62
string-matching	102	267	79.0	42.4	53.68	39.5	21.4	54.07	19.8	12.3	62.45
shortest-path	191	506	140.3	81.6	58.18	70.2	43.4	61.82	35.1	21.4	61.02
speech-1	247	652	187.2	108.6	58.03	93.6	54.0	57.70	46.8	29.9	63.79
speech-2	369	981	274.8	164.5	59.88	137.4	87.1	63.40	68.7	42.1	61.32
protein	546	1449	427.8	243.5	56.93	213.9	126.6	59.19	107.0	63.3	59.19
Average					54.75			53.44			52.08

These CNN applications are further partitioned based on the functionality (i.e., convolution, or pooling) to obtain CNN graphs. Our simulations are conducted based on the PIM architecture of Neurocube [8]. The Neurocube neuromorphic architecture is an extension of Micron's Hybrid Memory Cube to support neural computing. It provides 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 [6] and the proposed Para-CONV 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", "# of edge", and "IMP(%)" represent the number of convolution operations, the number of intermediate processing results, and the reduction of the total execution time over the baseline scheme SPARTA, respectively. From the experimental results, our approach can achieve an average 53.42% reduction, which corresponds to the acceleration of 1.87x in throughput.

2)Execution Time in Each Iteration: Para-CONV achieves the objective schedule by fully utilize all processing engines. This leads to the reduction of execution time in each iteration. Figure 5 illustrates the execution time of the application in each iteration. It is a loop kernel that will be repeatedly executed. The results are normalized by the execution time of the baseline scheme on 64 processing engines. Not surprisingly, the execution time in each iteration significantly decreases with more processing engines. This is due to the fact that additional processing engines can provide much benefit to let more convolutional connections execute in parallel.

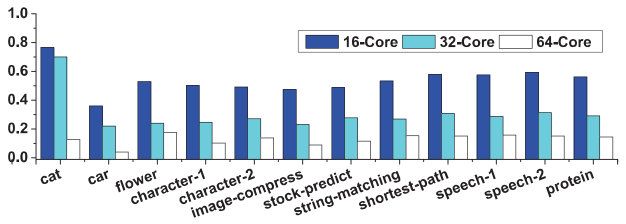

Figure 5: **The execution time in each iteration of our Para-CONV on 16, 32, and 64 processing elements.**

3)Prologue Time: Prologue time 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 Table 2. In Table 2, we report the maximum retiming value, as prologue time is $R_{max} \times p$. For all benchmarks, the maximum retiming value is sensitive to the number of processing engines. With the increasing of the number of processing engines, the maximum retiming value corresponding decreases. The scale of application also influences the maximum retiming value. Large scale applications will contribute to longer prologue time. Although several large-scale applications may introduce multiple iterations of prologue, compared to the benefit gained from the significant reduction of execution time in each iteration, this overhead is negligible.

4)Cache Allocation: Para-CONV aims to allocate intermediate processing results to appropriate location (on-chip cache or eDRAM) to improve the system throughput. Figure 6 illustrates the number of intermediate processing results that are allocated to on-chip cache of our Para-CONV on 16, 32, and 64 processing elements. From the results, for most of benchmarks, more intermediate processing results are allocated to on-chip cache when the configuration changes from 16 PEs to 32 PEs. However, the results for 32 PEs are quite the same as those for 64 PEs. This is because the convolutional connections in these benchmarks normally do not hold the concurrency with over thirty intermediate processing re-

Table 2: **The maximum retiming value of our Para-CONV on 16, 32, and 64 processing elements.**

	16-core	32-core	64-core	Average
cat	3	3	1	2.3
car	2	2	1	1.7
flower	3	2	2	2.3
character-1	6	3	2	3.7
character-2	7	5	3	5.0
image-compress	9	6	3	6.0
stock-predict	11	9	3	7.7
string-matching	14	8	5	9.0
shortest-path	24	13	8	15.0
speech-1	34	17	9	20.0
speech-2	49	27	16	30.7
protein	69	29	15	37.7

sults running in parallel. The parallelism of those benchmarks has been fully exploited with 32 PEs.

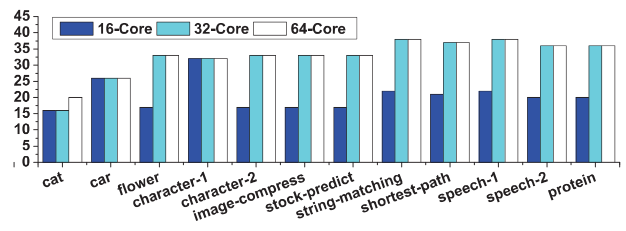

Figure 6: The number of intermediate processing results that are allocated to on-chip cache of our Para-CONV on 16, 32, and 64 processing elements.

5. CONCLUSION

This paper presents a data allocation framework, called Para-CONV, to minimize data movement for convolutional connections in processing-in-memory architecture. This is realized by exploiting parallelism for convolutions to fully utilize the processing engine, and finding the optimal allocation for intermediate processing results. We formulated the data movement minimization problem and solved it optimally using dynamic programming. Experiments using CNN configurations of various benchmarks show that the proposed Para-CONV can effectively reduce data movement and improve the throughput.

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 PIM architecture with CNN applications, which would provide potential opportunities of PIM-customized design.

Acknowledgments

The work described in this article is partially supported by the grants from National Natural Science Foundation of China (61502309), Guangdong Natural Science Foundation (2014A030310269, 2014A030313553 and 2016A030313045), Shenzhen Science and Technology Foundation (JCYJ20150525092941059 and JCYJ20150529164656096), Natural Science Foundation of SZU (701-000360055905, 701-00037134, and 827-000073), and State Key Laboratory of Computer Architecture, Institute of Computing Technology, Chinese Academy of Sciences (CARCH201608).

6. REFERENCES

[1] A. Graves et al. Hybrid computing using a neural network with dynamic external memory. *Nature*, 538(7626):471–476, Oct. 2016.

[2] 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 '16)*, pages 1–13, June 2016.

[3] 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 '16)*, pages 367–379, June 2016.

[4] 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 '16)*, pages 27–39, June 2016.

[5] J. Chung and T. Shin. Simplifying deep neural networks for neuromorphic architectures. In *2016 53nd ACM/EDAC/IEEE Design Automation Conference (DAC '16)*, pages 126:1–126:6, June 2016.

[6] 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 '16)*, pages 27:1–27:10, 2016.

[7] 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 '14)*, pages 10–14, Feb 2014.

[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 '16)*, pages 380–392, June 2016.

[9] J. L. Krichmar, P. Coussy, and N. Dutt. Large-scale spiking neural networks using neuromorphic hardware compatible models. *ACM Journal on Emerging Technologies in Computing Systems (JETC)*, 11(4):36:1–36:18, Apr. 2015.

[10] 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 '10)*, pages 253–256, May 2010.

[11] 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 '16)*, pages 1243–1248, March 2016.

[12] 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 '16)*, pages 173:1–173:6, June 2016.

[13] 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*, 43(7):507–519, Jul 1996.

[14] 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 '15)*, pages 1543–1546, March 2015.

[15] 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 '16)*, pages 123:1–123:6, June 2016.

[16] 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 '15)*, pages 1–9, June 2015.

[17] 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 '16)*, pages 18:1–18:6, June 2016.

[18] L. Xia, T. Tang, W. Huangfu, M. Cheng, X. Yin, B. Li, Y. Wang, and H. Yang. Switched by input: Power efficient structure for RRAM-based convolutional neural network. In *2016 53nd ACM/EDAC/IEEE Design Automation Conference (DAC '16)*, pages 125:1–125:6, June 2016.

[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 '15)*, pages 161–170, New York, NY, USA, 2015. ACM.