CNN Inference Workload Distribution Considering Accumulative Effect of Receptive Fields

I-Ting Ho, Li-Hsing Yen, and Yan-Wei Chen
Department of Computer Science, National Yang Ming Chiao Tung University, Hsinchu, Taiwan.

***Abstract*—Operating an artificial intelligence and machine learning (AI/ML) model requires significant computing power, posing a substantial challenge for edge devices with limited resources. A potential way to accelerate the operations is to distribute the workload to multiple collaborative devices. For the inference using a convolutional neural network (CNN), the workload distributed is complicated by the overlapping input tensors among workers due to the accumulative effect of receptive fields (AERF). This factor incurs extra communication time, which was overlooked by prior studies. In this paper, we propose an approach to CNN inference time minimization by aggregating the inference workload distributed to multiple heterogeneous computing devices. Our result shows a 10% speed-up against the benchmark.**

I. Introduction

Integrating AI/ML models with edge computing has given rise to a novel research topic known as edge intelligence [1]. Edge intelligence may refer to the intelligent edge, which uses AI/ML to assist resource management on the network edge, or edge intelligence, which supports the training and inference of AI/ML models on the network edge [2]. In particular, the latter direction involves aspects such as data collection and model offloading and is challenged by increasingly complex AI/ML models and resource-constrained edge devices. The trend of increasing model complexity demands more computing power, which causes longer computation time for lightweight edge devices. For this reason, many research efforts have been on accelerating AI/ML execution by distributing the workload to multiple servers on the network edge [1, 2, 3].

Edge servers can play an active role in various scenarios. This paper assumes the cloud-assisted, edge-hosted AI scenario [4], where cloud servers assist in training AI models while edge servers perform inference. Use cases of this scenario include automated driving and mobile virtual reality. We focus on accelerating techniques for inference with convolutional neural networks (CNNs) [5, 6]. Existing approaches to CNN inference acceleration root in three main concepts: model compression, early exiting, and model parallelism. Model compression trades model accuracy for inference speedup. Early exiting allows certain inference requests to stop inference midway to output results. Model parallelism overlaps the inference process among several computing devices, which is the main theme of this study.

The main idea of model parallelism is to allocate multiple computing devices to perform the inference process in parallel and merge the outputs for the final inference results. It consists of two parts: model partition and workload distribution. The former decomposes the inference model into multiple components, while the latter distributes the inference workload to computing devices. Although model parallelism reduces inference time, it also causes additional communication and waiting time due to inherent component dependency. An optimal solution should jointly consider computation, communication, and waiting time.

For CNNs, convolutional layers (CLs) are associated with massive floating-point operations, thus dominating overall computation time [7, 8]. *Layer-wise parallelism* distributes the inference workload of each individual CL into multiple parts to shorten the inference time [9] or improve energy consumption [10]. *Block-wise parallelism* fuse one or more CLs into a *fused block* [11]) and distribute the inference workload of the block to multiple devices to reduce the memory footprint [8] or the inference time [12, 11]. A critical issue of block-wise parallelism is *layer fusing*, i.e., to determine the set of CLs fused into each block.

In each CL, *kernel operation* gets the value of each entry in the output tensor via a dot product of a particular tensor called *kernel* and a patch of the same size in the input tensor. Since the output tensor of a CL is also the input tensor of the next CL, an entry in the input tensor of a CL depends on the values of a patch of the input tensor of the previous CL (known as a *receptive field* (RF) [13]). The scope of an RF amplifies when we trace back to an even earlier CL, meaning that the overlapping area of RFs expands as a fused block gets longer. This effect, referred to as *accumulative effect of RF* (AERF), increases the amount of a block's output tensors that a device needs to pull from other devices for its input and thus introduce additional communication and waiting time. However, prior studies [11, 12] did not specifically address this issue.

This paper proposes a block-wise CNN partition and workload distribution approach that considers communication and waiting time due to AERF to minimize the overall inference time. The main contributions of our work follow:

1) We revise the inference time formula to incorporate the impact of AERF.
2) We propose an algorithm that distributes the inference workload of a block to multiple heterogeneous devices to minimize the inference time of the block.
3) We propose a layer-fusing algorithm based on dynamic programming to minimize the inference time of a whole CNN.

We conducted experiments on multiple CNN models. The experimental results show that the proposed solution outper-

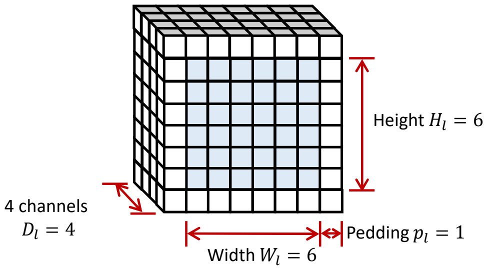


Figure 1: A CL with four feature maps (channels), each of which is a 6×6 tensor.

formed the previous work in [11] when devices differ significantly in computing power. In other scenarios, we achieved the same acceleration result.

The remainder of the paper is organized as follows. Sec. II provides background and an overview of related work. Sec. III formulates the problem. Sec. IV details the proposed algorithms for workload distribution and block-wise parallelism. Sec. V shows the experimental results. The last section concludes this paper.

II. Background and Related Work

A. CNN Model

A CNN model typically comprises a stack of interleaved convolutional layers (CLs) and pooling layers, with fully connected layers connected at the end. Let CL_l represent the l-th CL. The data input to CL_l consists of D_l equal-sized channels. Each channel is a feature map, i.e., a two-dimensional tensor with width W_l and height H_l. Hereafter, we assume a square feature map so $W_l = H_l$. A feature map is surrounded by a thin layer of padding pixels with width p_l, which are needed when performing kernel operations for data pixels. Fig. 1 shows an example of a CL with four 6×6 feature maps.

A kernel (also known as a filter or feature detector) for a single channel is a two-dimensional (width × height) tensor with values determined by training. We assume a $k_l \times k_l$ kernel for the input feature map of CL_l. A kernel for a CL becomes three dimensional when all kernels of each channel are combined. Kernel operations are fundamental to the functioning of CLs, which involve sliding a kernel over the input feature map and performing pixel-wise multiplication followed by summation to produce a single value in the output feature map. Another parameter called *stride* (s_l for CL_l) determines the step size (in pixels) when sliding the kernel. Kernel operations may reduce the dimension of each channel but do not change the number of channels in that layer. Since the output tensor of CL_l is the input tensor of CL_{l+1}, each output feature map of CL_l is of size $W_{l+1} \times H_{l+1}$. Fig. 2 shows an example of kernel operations in CL_l.

The relationship between H_{l+1} and H_l can be expressed as

$$H_{l+1} = \frac{(H_l - k_l + 2p_l)}{s_l} + 1. \tag{1}$$

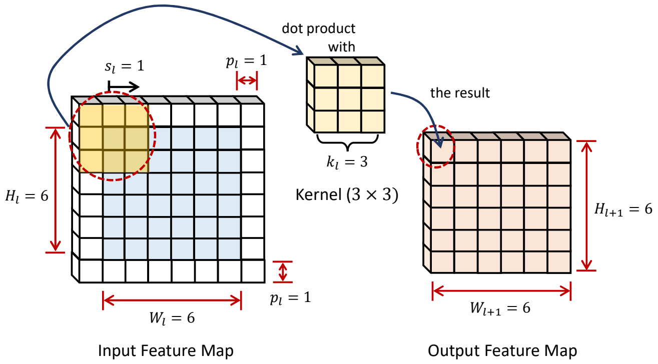


Figure 2: An example of kernel operations in CL_l.

For each entry (i.e., pixel) in the output tensor of CL_l, the number of scalar multiplications performed by the kernel operation is exactly the number of pixels in the kernel, which is $k_l^2 \times D_l$. Since there are D_{l+1} output channels, the number of floating-point multiplication operations (FLOPs) required for CL_l is [8], [11]:

$$k_l^2 \cdot D_l \cdot D_{l+1} \cdot W_{l+1} \cdot H_{l+1}. \tag{2}$$

B. Model Parallelism

A block-wise model parallelism fuses CLs into disjoint blocks for workload distribution. Each device is designated to produce a part of the output tensor of a block. For each output entry in CL_l that a device is supposed to produce, the device must get the contents of the corresponding RF. Let f_l be the width of the RF for any output entry in CL_l. We have the following equation for f_l [14]

$$f_l = \begin{cases} k_l, & \text{if } l = 1, \\ f_{l-1} + (k_l - 1) j_{l-1}, & \text{otherwise,} \end{cases} \tag{3}$$

where j_l represents the jump (the cumulative stride) in CL_l given by

$$j_l = \begin{cases} s_l, & \text{if } l = 1, \\ j_{l-1} s_l, & \text{otherwise.} \end{cases} \tag{4}$$

C. Related Work

Table I lists existing research on accelerating CNN inference by model parallelism. Some approaches were designed for layer-wise parallelism [9, 10]. Previous studies [12, 11] reported that when computation resource was the bottleneck, layer-wise parallelism reduced more computing time than block-wise parallelism. By contrast, when communication resource was the bottleneck, block-wise parallelism outperformed layer-wise parallelism.

Stahl et al. [8] considered block-wise parallelism and analyzed the impact of model partition on memory usage. However, they did not consider heterogeneous computing devices. Hou et al. [12] proposed a greedy layer-fusing algorithm together with a Deep Reinforcement Learning (DRL) approach to workload distribution for block-wise parallelism.

Table I: Related Work

Work	Heterogeneous devices	Layer fusing	Workload Distribution	AERF
[9]	Yes	No	Proportional	No
[10]	Yes	No	LP-solver	No
[8]	No	Yes	Equal	No
[12]	Yes	Greedy	DRL	No
[11]	Yes	DP	Proportional	No
This Work	Yes	DP	Greedy	Yes

Our work is mostly close to DPFP [11], which uses dynamic programming for layer fusing and an iterative algorithm for workload distribution. When allocating computing devices, DPFP picks up the one with the highest computational capacity first. It then checks to see if more devices should be allocated to reduce the inference time further. This process continues until no more devices are needed. Our work differs from [11] in considering AERF in inference time calculation.

III. Problem Formulation

We assume a set of N edge devices $\mathcal{N} = \{1, 2, ..., N\}$ with uniform link rate γ between each pair. Each device has enough memory capacity to execute a CNN model.

We assume a CNN model consisting of L CLs. Layer fusing in prior work [12, 11] fuses CLs into blocks. Such treatment does not prevent shortcuts across blocks, which demands extra inter-block data exchange. Our work redefines a block as the smallest possible set of consecutive CLs such that no shortcut between two CLs crosses a block boundary. Let $\mathcal{B} = \{1, 2, \cdots, B\}$ be the set of such blocks. We assume that each device has all the input data for the first block. We then perform *block grouping* to fuse blocks into groups. We use $\mathcal{G} = \{1, 2, ..., G\}$ to represent a set of G groups. Each group is non-empty and consists of one or more consecutive blocks. A possible block grouping for $\mathcal{G}$ can be characterized by a vector $\vec{\beta} = (\beta_1, \beta_2, ..., \beta_G)$, where β_g is the index of the ending block of group g for all $g \in \mathcal{G}$.

We represent a workload distribution for group g by $\vec{\alpha}_g = (\alpha_{g,1}, \alpha_{g,2}, ..., \alpha_{g,N}), \forall g \in \mathcal{G}$, where $\alpha_{g,n}$ is the number of rows (i.e., the height) of the output tensor in the last CL of group g that are allocated to device n. A specific workload distribution for $\mathcal{G}$ is represented by $\vec{\alpha} = (\vec{\alpha}_1, \vec{\alpha}_2, ..., \vec{\alpha}_G)$, where $\vec{\alpha}_g = (\alpha_{g,1}, \alpha_{g,2}, ..., \alpha_{g,N})$.

Let the computation and communication time of device n in group g be $T^{\text{comp}}_{g,n}$ and $T^{\text{comm}}_{g,n}$, respectively. Our problem is to determine the number of groups G, a block grouping $\vec{\beta}$, and a workload distribution for each group $\vec{\alpha}$ such that

$$\min_{G \in \mathcal{B}, \vec{\beta}, \vec{\alpha}} \sum_{g \in \mathcal{G}} \left(\max_{\forall n \in \mathcal{N}} \{T^{\text{comp}}_{g,n}\} + \sum_{\forall n \in \mathcal{N}} T^{\text{comm}}_{g,n} \right) \tag{5}$$

subject to $1 \leq \beta_1$, $\beta_g < \beta_{g+1}, \forall g \in \mathcal{G} \setminus \{G\}$, $\beta_G = B$, and

$$\sum_{n \in N} \alpha_{g,n} = H_{e_g+1}, \ \ \forall g \in \mathcal{G}, \tag{6}$$

where e_g is the index of the last CL in group g. (6) demands that all workload in each group should be fully distributed.

IV. Proposed Approach

We first show how to compute each device's computation and communication time for a given workload distribution in a group. We then propose two algorithms for our problem. Bottleneck-Improving Allocation (BIA) finds a workload distribution for a given group. It is time efficient but may end up with a local optimum. The other algorithm uses dynamic programming (DP) collaborating with BIA to find an optimal block grouping.

A. Computation Time Formulation

The computation time of a device is proportional to the number of FLOPs performed by the device [11, 8]. Let μ_n be the number of FLOPs per second (FLOPS) device n can perform. The computation time of device n in group g is

$$T^{\text{comp}}_{g,n} = \frac{F_{g,n}}{\mu_n}, \forall n \in \mathcal{N}, \forall g \in \mathcal{G}, \tag{7}$$

where $F_{g,n}$ is the number of FLOPs performed by device n in group g. The value of $F_{g,n}$ is determined by the number of consecutive rows in the output tensor of group g (more explicitly, the last CL of group g) allocated to device n.

For any CL l in group g, let $\text{OS}^l_{g,n}$ and $\text{OE}^l_{g,n}$ denote the indices of the starting and the ending rows, respectively, of the output tensor allocated to device n. $\text{OS}^l_{g,n}$ and $\text{OE}^l_{g,n}$ determine $H_{l+1,n}$, the height of the output tensor allocated to device n in the CL, as

$$H_{l+1,n} = \begin{cases} 0, & \text{if } \alpha_{g,n} = 0, \\ \text{OE}^l_{g,n} - \text{OS}^l_{g,n} + 1, & \text{otherwise.} \end{cases} \tag{8}$$

We compute $\text{OS}^l_{g,n}$ and $\text{OE}^l_{g,n}$ sequentially from the last CL to the preceding CLs, progressing backward through the group. For the last CL in group g denoted by e_g, it is easy to see that $\text{OS}_{g,n} = \sum_{k=1}^{n-1} \alpha_{g,k} + 1$ and $\text{OE}_{g,n} = \sum_{k=1}^{n} \alpha_{g,k}$. We can follow the definition of the RFs to compute $\text{OS}^l_{g,n}$ and $\text{OE}^l_{g,n}$ for other $l \neq e_g$. It turns out that $\text{OS}^l_{g,n} = \max\{1, 1 - p_{l+1} + (\text{OS}^{l+1}_{g,n} - 1) \cdot s_{l+1}\}$ and $\text{OE}^l_{g,n} = \min(H_{l+1}, (1 - p_{l+1} + \lfloor (k_{l+1} - 1)/2) \rfloor) + (\text{OE}^{l+1}_{g,n} - 1) \cdot s_{l+1} + \lfloor (k_{l+1} - 1)/2) \rfloor)$.

By (2), we have

$$F_{g,n} = \sum_{l=e_{g-1}+1}^{e_g} k_l^2 \cdot D_l \cdot D_{l+1} \cdot W_{l+1} \cdot H_{l+1,n}. \tag{9}$$

B. Communication Time Formulation

Because of the model partition, each device may not have all the required input data and must pull the missing input data from other devices. This activity causes communication time, which is equal to the number of bytes transmitted divided by the transmission rate.

No communication time is needed for the first block because each device has the entire input tensor. Therefore, $T^{\text{comm}}_{1,n} = 0$ for all $n \in \mathcal{N}$. For any other group $g \neq 1$, each device only receives a range of the input tensor. We can calculate the range by tracing back the RF areas starting from the part of the output tensor of the same group that is allocated to the device.

Let $\mathrm{RD}_{g,n}$ be the number of rows in the input tensor of group g but not in the output tensor of group $g-1$ that is allocated to device n. Assuming a 32-bit floating-point number for each pixel, the communication time for device n in group g is

$$T^{\text{comm}}_{g,n} = \frac{4 \cdot D_{e_{g-1}+1} \cdot W_{e_{g-1}+1} \cdot \mathrm{RD}_{g,n}}{\gamma}, \forall n \in \mathcal{N}. \quad (10)$$

We derive the value of $\mathrm{RD}_{g,n}$ as follows. Let $\mathrm{IS}_{g,n}$ and $\mathrm{IE}_{g,n}$ denote the indices of the starting and ending rows, respectively, of the input tensor allocated to device n in group g. The values of $\mathrm{IS}_{g,n}$ and $\mathrm{IE}_{g,n}$ can be calculated by $\mathrm{OS}_{g,n}$, $\mathrm{OE}_{g,n}$ and RF [11]:

$$\mathrm{IS}_{g,n} = \max(1, \sigma_{e_g} + (\mathrm{OS}_{g,n} - 1) j_{e_g} - \lfloor (f_{e_g} - 1)/2 \rfloor), \quad (11)$$

$$\mathrm{IE}_{g,n} = \min(H_{e_{g-1}+1}, \sigma_{e_g} + (\mathrm{OE}_{g,n} - 1) j_{e_g} + \lfloor (f_{e_g} - 1)/2 \rfloor). \quad (12)$$

The definitions of f_l and j_l are the same as in (3) and (4) except that $f_l = k_l$ and $j_l = s_l$ when $l = e_{g-1} + 1$ (i.e., the first CL in group g). The value of σ_l is as follows.

$$\sigma_l = \begin{cases} 1 - p_l + \lfloor (k_l - 1)/2 \rfloor, & \text{if } l = e_{g-1} + 1, \\ \sigma_{l-1} + (\lfloor (k_l - 1)/2 \rfloor - p_l) j_{l-1}, & \text{otherwise.} \end{cases} \quad (13)$$

Let $\mathrm{OS}_{g,n}$ and $\mathrm{OE}_{g,n}$ represent the starting and ending rows, respectively, of the output tensor of group g that are allocated to device n. We have

$$\mathrm{RD}_{g,n} = \max(\mathrm{OS}_{g-1,n} - \mathrm{IS}_{g,n}, 0) + \max(\mathrm{IE}_{g,n} - \mathrm{OE}_{g-1,n}, 0). \quad (14)$$

C. Bottleneck Improving Allocation (BIA)

The workload for distribution in group g is the total number of rows in the output tensor, H_{β_g+1}. Workload distribution determines the inference time of a device. The inference time of device n in group g is the sum of ($T^{\text{comm}}_{g,n}$) and ($T^{\text{comp}}_{g,n}$). A device with the largest inference time in group g represents the bottleneck of the current allocation. BIA shown in Algorithm 1 initially allocates all rows to the device with the largest computation capacity. It then iteratively attempts to improve the inference time by shifting one workload unit (i.e., one row of the output tensor) from the bottleneck device to the one with the shortest inference time. Time complexity of BIA is $\mathrm{O}(H_{\beta_g+1} \cdot N)$.

D. Block Grouping Algorithm

The block grouping algorithm determines whether each block should be in the same group (as the previous block) or start a new one. Therefore, there are 2^B possible ways of block grouping. In the case when every block is a group, the solution space of workload distribution is $\prod_{i\in\mathcal{B}} \binom{H_{e_i+1}+N-1}{N}$ [15].

As we were inspired by [11], an optimal grouping is the optimal first group followed by the optimal grouping for the remaining set of blocks. We use Algorithm 2 to decide a new group from the remaining blocks. The newly created group will invoke BIA to calculate an inference time. Algorithm 3 finds an optimal grouping by checking all possible grouping solutions. It works like a variant of matrix chain multiplication.

Algorithm 1 Bottleneck-Improving Allocation (BIA)

Input: CNN model C, computation capacity $\{\mu_n\}_{n\in\mathcal{N}}$, ending convolution layer β_g starting convolution layer $\beta_{g-1}+1$, workload distribution of previous group $\vec{\alpha}_{g-1}$

Output: workload distribution $\vec{\alpha}_g$ and bottleneck time

```
procedure BIA($\beta_{g-1}+1, \beta_g, \vec{\alpha}_{g-1}$)
    $\vec{\alpha_g} \leftarrow (0, 0, \cdots, 0)$
    $\bar{n} \leftarrow \arg\max_{n\in\mathcal{N}}\{\mu_n\}$
    $\alpha_{g,\bar{n}} \leftarrow H_{e_{\beta_g}+1}$
    Calculate $T^{\text{comp}}_{g,\bar{n}}$ by (7)
    Calculate $T^{\text{comm}}_{g,\bar{n}}$ by (10)
    repeat
        $T^{\max}_g \leftarrow T^{\text{comp}}_{g,\bar{n}} + T^{\text{comm}}_{g,\bar{n}}$.
        $T^{\text{opt}}_g \leftarrow T^{\max}_g$
        for each device $n \in \mathcal{N} \setminus \{\bar{n}\}$ do
            $\vec{\alpha_g}' \leftarrow \vec{\alpha_g}$ with 1 row shifted from $\bar{n}$ to $n$
            Calculate $T^{\text{comp}}_{g,n}$ according to $\vec{\alpha_g}'$
            Calculate $T^{\text{comm}}_{g,n}$ according to $\vec{\alpha_g}', \vec{\alpha}_{g-1}$
            if $T^{\text{comp}}_{g,n} + T^{\text{comm}}_{g,n} < T^{\text{opt}}_g$ then
                $T^{\text{opt}}_g \leftarrow T^{\text{comp}}_{g,n} + T^{\text{comm}}_{g,n}$
                $k \leftarrow n$
            end if
        end for
        if $T^{\text{opt}}_g < T^{\max}_g$ then
            Update $\vec{\alpha_g}$ by shifting 1 row from $\bar{n}$ to $k$
            Update $T^{\text{comp}}_{g,n}$ and $T^{\text{comm}}_{g,n}$ for all $n \in \mathcal{N}$
            $\bar{n} \leftarrow \arg\max_{n\in\mathcal{N}}(T^{\text{comp}}_{g,n} + T^{\text{comm}}_{g,n})$
        end if
    until $T^{\text{opt}}_g \geq T^{\max}_g$
    return $(\vec{\alpha}_g, T^{\text{comp}}_{g,\bar{n}} + T^{\text{comm}}_{g,\bar{n}})$
end procedure
```

Algorithm 2 Best Partition Point (BPP)

Input: indices of starting and ending blocks: i and j; workload distribution of block $i-1$: $\vec{\alpha}'$

Output: inference time t^*

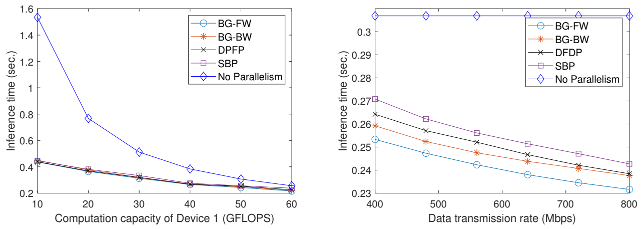
```
procedure BPP($i, j, \vec{\alpha}'$)
    if $\exists t$ such that $(\{i, j, \vec{\alpha}'\}, t) \in \mathcal{T}$ then
        return $t$
    end if
    $t^* \leftarrow \infty$.
    $\beta \leftarrow i$
    for each $k$ in $[i, j]$ do
        $(\vec{\alpha}, t_{i,k}) \leftarrow$ BIA$(i, k, \vec{\alpha}')$
        $t_{k+1,j} \leftarrow \begin{cases} 0, & \text{if } k+1 > j, \\ \text{BPP}(k+1, j, \vec{\alpha}), & \text{otherwise.} \end{cases}$
        if $t_{i,k} + t_{k+1,j} < t^*$ then
            $t^* \leftarrow t_{i,k} + t_{k+1,j}$
            $\beta \leftarrow k$
        end if
    end for
    add $(\{i, j, \vec{\alpha}'\}, \beta)$ to $\mathcal{P}$
    add $(\{i, j, \vec{\alpha}'\}, t^*)$ to $\mathcal{T}$
    return $t^*$
end procedure
```

V. Numerical Results

We performed experiments to evaluate the performance of our approach in comparison with those of the following four alternatives. a) DPFP [11]. b) BG-BW, a variant of BG-FW that performs block grouping backward. c) Single-Block Parallelism (SBP), a degraded version of DPFP that performs workload distribution assuming all single-block groups. d) No parallelism, which executes the entire CNN model on the most

Algorithm 3 Block Grouping: Forward (BG-FW)

```
    Input: CNN model $C$; Computation capacity $\{\mu_n\}_{n\in\mathcal{N}}$
    Output: Grouping strategy $\vec{\beta}$
procedure BG-FW($C, \{\mu_n\}_{n\in\mathcal{N}}$)
    $\mathcal{T} \leftarrow \emptyset$                    ▷ Hash table to keep inference time
    $\mathcal{P} \leftarrow \emptyset$                    ▷ Hash table to keep grouping result
    $(\vec{\alpha}, t_{1,1}) \leftarrow$ BIA$(1, 1, \vec{\alpha}')$
    $t \leftarrow$ BPP$(1, B, \vec{\alpha})$                              ▷ get best time
    $i \leftarrow 1$; $g \leftarrow 1$
    while $i \leq B$ do                        ▷ loop to collect grouping results
        get $(\{i, B, \vec{\alpha}\}, y)$ from $\mathcal{P}$
        $\beta_g \leftarrow y$
        $i \leftarrow y + 1$; $g \leftarrow g + 1$
    end while
end procedure
```



(a) Effect of computation capacity

(b) Effect of transmission rate

Figure 3: Results with VGG16 (Default: $\gamma = 560$ Mbps, $(\mu_1, \mu_2, \mu_3, \mu_4) = (50, 10, 10, 10)$ GFLOPS)

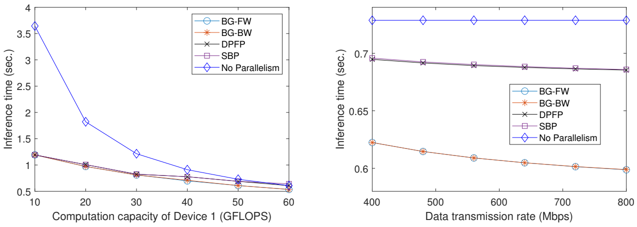
powerful device. We assumed four computing devices, where Device 1 had five times the computational capacity (in FLOPS) of any others. The default transmission data rate γ is 560 Mbps. We tested three popular CNN models: (a) VGG16 [16] (b) Resnet-34 [17] (c) Darknet-53 [18] in the experiments.

A. *Scenario:VGG16*

For VGG16, each block consists of only one CL since there is no shortcut between CLs. Fig. 3a shows how the inference time decreased as device computation capacities increased. No Parallelism relied on the most powerful device (Device 1) to perform the inference alone and thus had a longer inference time than any other approach. However, the performance gap shrank when Device 1 became powerful enough to perform the inference alone. With 60 GFLOPS, No Parallelism performed even better than SBP due to the extra communication time between devices in SBP.

By contrast, all approaches, excluding No Parallelism, benefited from increased transmission rate (Fig. 3b), but the proposed approach outperformed all others due to the consideration of waiting and communication time. DPFP had a shorter inference time than SBP because DPFP additionally considered layer fusing to further reduce communication time.

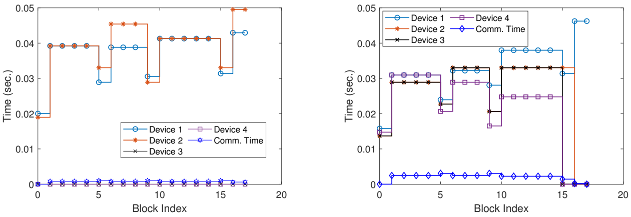
BG-FW outperforms BG-BW due to the inherent structure of CNN. In CNNs, the early layers are more amenable to parallelism, and the impact of AERF is relatively minimal. Conversely, the later layers exhibit a reversed pattern. BG-BW initially prioritizes parallel execution for the later layers, but this approach overlooks the fact that excessive parallelism may not be ideal for these layers. As a result, BG-BW performed worse than BG-FW.



(a) Effect of computation capacity

(b) Effect of transmission rate

Figure 4: Results with Resnet-34 (Default: $\gamma = 560$ Mbps, $(\mu_1, \mu_2, \mu_3, \mu_4)$= $(5, 1, 1, 1)$ GFLOPS)



(a) DPFP

(b) BG-FW

Figure 5: Computation time of each device and total communication time in (a) DPFP and (b) BG-FW

B. *Scenario:Resnet-34*

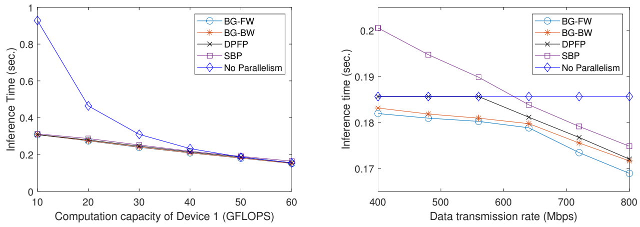
For Resnet-34, a block might consist of more than one CL due to shortcuts across CLs. Fig. 4a shows how the inference time decreased with increased device computation capacities, which repeats the same trend as shown in Fig. 3a. When the computation capacities were lower, inference time benefited from parallel inference.

Fig. 4b shows the impact of transmission rate on inference time. The performance gap between BG-FW and DPFP was larger than that shown in Fig. 3b. This is because a block in Resnet-34 might contain multiple CLs, and thus, a device generally needs to pull more data from other devices due to the AERF. Because DPFP overlooked this communication overhead, it distributed the workload to only two devices for parallel inference (Fig. 5a). By contrast, BG-FW allocated four devices for inference (Fig. 5b), resulting in an overall small inference time at the cost of a slightly higher communication time. BG-FW and BG-BW behaved similarly, which suggests that the direction of block grouping did not matter for Resnet-34.

C. *Scenario:Darknet-53*

Like Resnet-34, a block in Darknet-53 might consist of more than one CLs due to shortcuts across CLs. Fig. 6a shows the impact of computation capacity on inference time, which is similar to the results of the other models.

Fig. 6b shows the impact of transmission rate on inference time. When the transmission rates were low, No Parallelism performed better than SBP because it incurred no communication time, and DPFP performed identically to No Parallelism



(a) Effect of computation capacity

(b) Effect of transmission rate

Figure 6: Results with Darknet-53 (Default: $\gamma = 560$ Mbps, $(\mu_1, \mu_2, \mu_3, \mu_4) = (50, 10, 10, 10)$ GFLOPS)

because DPFP also incurred no communication time by strategically placing all workloads on a single device. BG-FW and BG-BW outperformed DPFP due to the consideration of the AERF and the use of more devices. The performance gap was smaller than that for Resnet-34 because Darknet-53 had fewer channels and a smaller kernel size in the starting layer.

VI. Conclusions

We have proposed a block-wise CNN partition and workload distribution approach that considers AERF to minimize the overall inference time. The approach uses dynamic programming to fuse blocks into groups and attempts to improve inference time in each group by iteratively adjusting one unit of workload. Simulation results show that the proposed approach performed better than its counterparts, particularly when more devices with lower computing power were involved. The improvement ratio depends on the CNN architecture. Further work is needed to optimize our algorithm for more complex AI/ML models such as Transformer, which is more intricate than CNN but still shares some common properties such as AERF.

Acknowledgment

This work was supported by the National Science and Technology Council of Taiwan under grant number NSTC 113-2218-E-A49-027.

References

[1] S. Deng, H. Zhao, W. Fang, J. Yin, S. Dustdar, and A. Y. Zomaya, “Edge intelligence: The confluence of edge computing and artificial intelligence,” *IEEE Internet Things J.*, vol. 7, pp. 7457–7469, Aug. 2020.

[2] X. Wang, Y. Han, V. C. Leung, D. Niyato, X. Yan, and X. Chen, “Convergence of edge computing and deep learning: A comprehensive survey,” *IEEE Commun. Surveys Tuts.*, vol. 22, pp. 869–904, Apr. 2020.

[3] D. Xu, T. Li, Y. Li, X. Su, S. Tarkoma, T. Jiang, J. Crowcroft, and P. Hui, “Edge intelligence: Empowering intelligence to the edge of network,” *Proc. IEEE*, vol. 109, pp. 1778–1837, Nov. 2021.

[4] M. Li, J. Gao, C. Zhou, X. Shen, and W. Zhuang, “Slicing-based artificial intelligence service provisioning on the network edge,” *IEEE Veh. Technol. Mag.*, vol. 16, no. 4, pp. 16–26, Dec. 2021.

[5] L. Alzubaidi, J. Zhang, A. J. Humaidi, A. Al-Dujaili, Y. Duan, O. Al-Shamma, J. Santamaría, M. A. Fadhel, M. Al-Amidie, and L. Farhan, “Review of deep learning: concepts, CNN architectures, challenges, applications, future directions,” *J. Big Data*, vol. 8, pp. 1–74, Jun. 2021.

[6] Z. Li, F. Liu, W. Yang, S. Peng, and J. Zhou, “A survey of convolutional neural networks: Analysis, applications, and prospects,” *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 33, pp. 6999–7019, Dec. 2022.

[7] Z. Zhao, K. M. Barijough, and A. Gerstlauer, “DeepThings: Distributed adaptive deep learning inference on resource-constrained IoT edge clusters,” *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, vol. 37, pp. 2348–2359, Nov. 2018.

[8] R. Stahl, A. Hoffman, D. Mueller-Gritschneder, A. Gerstlauer, and U. Schlichtmann, “DeeperThings: Fully distributed CNN inference on resource-constrained edge devices,” *Int. J. Parallel Program.*, vol. 49, pp. 600–624, Aug. 2021.

[9] T. Mohammed, C. Joe-Wong, R. Babbar, and M. D. Francesco, “Distributed inference acceleration with adaptive DNN partitioning and offloading,” in *Proc. IEEE INFOCOM*, 2020, pp. 854–863.

[10] L. Zeng, X. Chen, Z. Zhou, L. Yang, and J. Zhang, “CoEdge: Cooperative DNN inference with adaptive workload partitioning over heterogeneous edge devices,” *IEEE/ACM Trans. Netw.*, vol. 29, pp. 595–608, Apr. 2021.

[11] N. Li, A. Iosifidis, and Q. Zhang, “Collaborative edge computing for distributed CNN inference acceleration using receptive field-based segmentation,” *Comput. Netw.*, vol. 214, Sep. 2022.

[12] X. Hou, Y. Guan, T. Han, and N. Zhang, “DistrEdge: Speeding up convolutional neural network inference on distributed edge devices,” in *Proc. IEEE IPDPS*, May 2022, pp. 1097–1107.

[13] W. Luo, Y. Li, R. Urtasun, and R. Zemel, “Understanding the effective receptive field in deep convolutional neural networks,” in *Proc. NIPS’16*, 2016, p. 4905–4913.

[14] “Receptive field calculations for convolutional neural networks,” https://rubikscode.net/2021/11/15/receptive-field-arithmetic-for-convolutional-neural-networks/, accessed: 2024-05-8.

[15] L. H. Yen, C. Y. Cheng, and Y. Su, “State transition reduction for sleep scheduling in IEEE 802.16e mobile subscriber stations,” *Telecommun. Syst.*, vol. 53, pp. 247–261, Jun. 2013.

[16] K. Simonyan and A. Zisserman, “Very deep convolutional networks for large-scale image recognition,” 2014, arXiv:1409.1556.

[17] K. He, X. Zhang, S. Ren, and J. Sun, “Deep residual learning for image recognition,” in *Proc. IEEE/CVF CVPR*, 2016, pp. 770–778.

[18] J. Redmon and A. Farhadi, “Yolov3: An incremental improvement,” 2018, arXiv:1804.02767.