Partitioning Deep Neural Networks for Optimally Pipelined Inference on Heterogeneous IoT Devices with Low Latency Networks

Woobean Seo[1], Saehwa Kim[2], and Seongsoo Hong[1]
Department of Electrical and Computer Engineering, *Seoul National University,* Seoul, Republic of Korea[1]
Department of Information Communications Engineering, *Hankuk-University of Foreign Studies,* Yongin, Republic of Korea[2]
wbseo@redwood.snu.ac.kr, ksaehwa@hufs.ac.kr, and sshong@redwood.snu.ac.kr

Abstract—**Pipeline parallelization is an effective technique that enables the efficient execution of deep neural network (DNN) inference on resource-constrained IoT devices. To support pipeline parallelization on heterogeneous computing nodes with low-latency networks, we propose DNNPipe, a DNN partitioning algorithm that constructs a pipeline plan for a given DNN. The primary objective of DNNPipe is to maximize the throughput of DNN inference while minimizing the runtime overhead of DNN partitioning, which is repeatedly executed in dynamically changing IoT environments. To achieve this, DNNPipe uses dynamic programming for an exhaustive exploration to find the optimal pipeline plan whose maximum stage execution time is no greater than that of any other possible pipeline plan. Additionally, it aggressively prunes suboptimal pipeline plans using an upper bound on the minimum value among all possible pipeline plans' maximum stage execution times. Experimental results demonstrate that DNNPipe significantly reduces its execution time and iteration counts compared to PipeEdge, the fastest known optimal DNN partitioning algorithm.**

Keywords—IoT, DNN, embedded AI, pipeline parallelization, DNN partitioning.

I. Introduction

The proliferation of Internet of Things (IoT) technology and the adoption of AI in various IoT applications has led to an increased demand for real-time processing of streaming data on deep neural networks (DNN). However, the limited processing power of IoT devices often poses a challenge to the efficient execution of even simple AI models, resulting in high latency and low throughput. To address this challenge, pipeline parallelization has emerged as a promising solution that enables the efficient execution of DNN inference on resource-constrained IoT devices.

Pipeline parallelization relies on DNN partitioning, which divides DNN layers into multiple stages and distributes their workload across multiple heterogeneous computing nodes. It can accelerate DNN inference through parallel and pipelined execution.

A DNN partitioning algorithm needs to maximize the throughput of the DNN inference while minimizing its runtime overhead. Note that the throughput of pipelined DNN inference is determined by the pipeline stage with the maximum execution time.

Many optimal DNN partitioning algorithms that maximize the throughput of DNN inference have been actively investigated in [1], [2], but they have high time complexity due to their reliance on brute-force algorithms. PipeEdge [3], which is known as the fastest optimal DNN partitioning algorithm, uses dynamic programming (DP) to reduce its time complexity. However, it still has a high time complexity, incurring a significant runtime overhead.

In this paper, we propose a DP-based optimal DNN partitioning algorithm, which we call DNNPipe. It divides an input DNN into a series of pipeline stages and assigns each stage to a specific computing node. The resulting partitioned DNN model, together with the allocation of its stages to computing nodes is referred to as a *pipeline plan*.

Although an optimal DNN partitioning algorithm tends to be computationally intensive, minimizing its runtime overhead is crucial to ensure efficient execution, as the dynamic nature of an IoT environment, where devices frequently join and leave the network, requires frequent re-execution of DNN partitioning.

DNNPipe efficiently prunes suboptimal pipeline plans by using an upper bound on the minimum value among all possible pipeline plans' maximum stage execution times. Our experimental results show that DNNPipe significantly reduces its execution time and iteration counts compared to PipeEdge.

II. Problem Formulation

We consider a system with N heterogeneous devices connected by a low-latency network. We assume a network in which the maximum communication latency between adjacent stages is less than the maximum stage execution time. This allows us to focus on optimizing only the computation time of each stage. We define a DNN model as a sequence of layers, where each layer can be either a single layer or a block of layers, such as a transformer block, and both are treated as a unit of partitioning.

Let $C = \{c_1, c_2, \ldots, c_N\}$ represent the set of the performance scaling factors of N devices relative to device 1. Similarly, let $E = \{e_1, e_2, \ldots, e_L\}$ denote the set of execution times of L layers when each layer is executed on device 1. With this, the execution time of layer i, when running on device j, is e_i/c_j.

A pipeline plan P^k is a sequence of stages $\langle \sigma_1, \sigma_2, \ldots, \sigma_i, \ldots, \sigma_{|P^k|} \rangle$. We also represent stage σ_i with a tuple (f_i, l_i, d_i) which dictates that σ_i executes the layers from f_i

to l_i inclusively on device d_i. The execution time of stage σ_i is denoted by $ST(\sigma_i) = ST(f_i, l_i, d_i) = \sum_{j=f_i}^{l_i} e_j / c_{d_i}$.

We further denote the maximum stage execution time of P^k by MST^k. By definition, the optimal pipeline plan is the plan whose maximum stage execution time is no greater than that of any other possible pipeline plan. We denote the optimal pipeline plan and its maximum stage execution time by P^* and MST^*, respectively.

III. Solution Approach

The key idea behind DNNPipe is to derive an upper bound on MST^* and prune a pipeline plan that has a stage whose execution time is greater than this upper bound. We derive the upper bound as follows:

$$\widehat{MST}^* = \frac{\Sigma_{e_i \in E}(e_i)}{\Sigma_{c_j \in C}(c_j)} + \frac{\max_{e_i \in E}(e_i)}{\min_{c_j \in C}(c_j)} \quad (1)$$

We let $IST = \Sigma_{e_i \in E}(e_i) / \Sigma_{c_j \in C}(c_j)$. Note that IST represents the share of the workload that is assigned to device 1. If we distribute this share to device j in proportion to its performance scaling factor c_j, each device will have the same execution time IST as device 1. Thus, IST is the most balanced ideal stage execution time.

The IST is an ideal value in the sense that it assumes a single layer can be divided between two adjacent devices. In reality, a device either contains a whole layer or not at all. This causes a difference between a stage's possible execution time and IST, and the amount of such difference is bounded by $\max_{e_i \in E}(e_i) / \min_{c_j \in C}(c_j)$. By adding this bound to IST, we have $\widehat{MST}^*$. We prove that $\widehat{MST}^*$ is a proper upper bound on MST^*, as follows.

Theorem 1. $\widehat{MST}^* \geq MST^*$

Proof: We show that $\exists P^k$ such that $MST^k \leq \widehat{MST}^*$. For any DNN, we can make $P^k = \langle \sigma_1, \sigma_2, \ldots, \sigma_{|P^k|} \rangle$ while assigning each stage σ_i such that $ST(f_i, l_i, d_i) - e_{l_i} / c_{d_i} \leq IST < ST(f_i, l_i, d_i)$. The last stage $\sigma_{|P^k|}$ may not be satisfied, in which case it has $ST(\sigma_{|P^k|}) < IST$ because $\Sigma_{\sigma_i \in P^k}\{c_{d_i} \cdot ST(\sigma_i)\} = \Sigma_{e_j \in E}(e_j) = IST \cdot \Sigma_{c_m \in C}(c_m)$. Note that if $ST(\sigma_{|P^k|}) \geq IST$, $\Sigma_{\sigma_i \in P^k}\{c_{d_i} \cdot ST(\sigma_i)\} > \Sigma_{e_i \in E}(e_i)$. Since $e_{l_i} / c_{d_i} \leq \max_{e_j \in E}(e_j) / \min_{c_l \in C}(c_l)$, for all σ_i in P^k, $ST(\sigma_i) \leq IST + \max_{e_j \in E}(e_j) / \min_{c_m \in C}(c_m) = \widehat{MST}^*$. Since $MST^k \leq \widehat{MST}^*$ and $MST^* \leq MST^k$, it follows that $MST^* \leq \widehat{MST}^*$.

Algorithm 1 shows the pseudocode of DNNPipe. For a given E and C, DNNPipe first calculates $\widehat{MST}^*$. It uses the table DP to store the intermediate results for dynamic programming. To explore all possible device configurations that could be involved in pipelining, DNNPipe iterates over a power set, which is a set of all subsets of N devices ordered by increasing cardinality. Note that DNNPipe extends each plan one stage at a time, rather than generating each possible plan in its entirety.

Algorithm 1: DNNPipe

```
Input: N, L, C, E
Output: The optimal pipeline plan P*
1  $\widehat{MST}^*$ ← sum(E)/sum(C) + max(E)/min(C) // by Eq. (1)
2  DP ← initialize_table(N, L); // create and initialize DP table
3  foreach (∀S, S ∈ ordered_power_set({1, 2, …, N})) {
4    foreach (∀k, k ∈ {1,2, …, N} − S) {
5      foreach (∀i, 1 ≤ i ≤ L) {
6        foreach (∀j, i ≤ j ≤ L){
7          if (ST(i, j, k) > $\widehat{MST}^*$) break; // to apply $\widehat{MST}^*$ by Theorem 1
8          update(DP, S, i, j, k, ST(i, j, k)); // update the DP table
9  }}}} // end-if/foreach
10 P* ← get_optimal_plan(DP); // get optimal pipeline plan from DP table
11 return P*
```

Each partially extended plan represents the plan with the minimum of the maximum stage execution times among partial plans to date when layers 1 to $i-1$ are pipelined with devices in S. DNNPipe then assigns the next stage with layers i to j to device k, which has not yet been assigned, and calculates $ST(i, j, k)$. If this exceeds $\widehat{MST}^*$, the plan is pruned.

IV. Experimental Results

We have performed experiments to compare DNNPipe with PipeEdge. We randomly generated E and C with a uniform distribution in the range of [50, 250] and [0.1, 2.0], respectively. We ran the experiments with 100 random E and C values, varying the parameters as follows: (1) N from 3 to 9 with L fixed at 300, and (2) L from 50 to 400 with N fixed at 8.

DNNPipe reduces execution times by 33% to 72% and iteration counts by 47.4% to 73.4% compared to PipeEdge, with the performance gap widening as N and L increase.

V. Conclusion

We proposed DNNPipe, an efficient and optimal DP-based DNN partitioning algorithm. The key contribution lies in DNNPipe's aggressive pruning strategy, which exploits an upper bound on the minimum value among all possible pipeline plans' maximum stage execution times. This allows us to prune suboptimal plans early and focus on the exploration in promising regions of the solution space. Our experimental results show that DNNPipe significantly reduces execution time compared to the fastest known optimal DNN partitioning algorithm.

Acknowledgement

This work was supported by Institute of Information & communications Technology Planning & Evaluation (IITP) grant funded by the Korea government (MSIT) (No.2021-0-00697, Development of Validation Technology for Operation Rights SW Safety and Response according to Fallback MRC of Edge-Based Autonomous Driving Function).

References

[1] X. Chen, et al., "DNNOff: Offloading DNN-Based Intelligent IoT Applications in Mobile Edge Computing," IEEE Transactions on Industrial Informatics, vol. 18, no. 4, 2ST022.

[2] M. Xu, et al., "DeepWear: Adaptive local offloading for on-wearable deep learning," IEEE Transactions on Mobile Computing, vol. 19, no. 2, 2020. for CNNs Leveraging Apache TVM," IEEE Access, vol. 11, 2023.

[3] Y. Hu, et al., "PipeEdge: Pipeline Parallelism for Large-Scale Model Inference on Heterogeneous Edge Devices," Proc. of Euromicro Conference on Digital System Design, 2022.