OverSeer: Efficient DNN Execution on Heterogeneous Devices

Rohan Mukherjee[1,2], Zhi Chen[1], Vin Sharma[1], Animesh Jain[1]
[1]Amazon Web Services, [2]Rice University
{mukrohan,chzhi,vinarm,janimesh}@amazon.com

Abstract

Deep learning applications have become pervasive in our daily lives, and are being executed on a wide variety of heterogeneous devices. This paper presents *OverSeer*, a topology- and hardware-agnostic end-to-end system, that uses a novel partitioning algorithm to efficiently utilize different computational capabilities present in heterogeneous devices.

ACM Reference Format:
Rohan Mukherjee[1,2], Zhi Chen[1], Vin Sharma[1], Animesh Jain[1]. 2019. *OverSeer*: Efficient DNN Execution on Heterogeneous Devices. In *Proceedings of ACM Conference (Workshop on AI Systems at SOSP'19)*. ACM, New York, NY, USA, 3 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 Introduction

Deep learning has made significant breakthroughs in solving complex problems, that only humans were considered good at [2, 3, 6, 9]. We are interacting with many user-facing applications on a daily basis that use Deep Neural Networks (DNN) underneath. Due to widespread DNN adoption, DNN execution happens on a wide variety of heterogeneous hardware devices [1] [8]. Applications requiring massive compute and memory are offloaded to datacenter servers. We are also witnessing a trend of edge-device DNN execution, driven primarily by latency and privacy concerns [5].

In such diverse heterogeneous environments, it becomes imperative to design a technique that can efficiently utilize the heterogeneous device capabilities. For example, we observe that we can lose 10% performance for *ResNet-152* on an HP desktop *x86*-platform, with CPU and an integrated graphics card, in the absence of an intelligent partitioner. Most of the relevant prior work deals with exploiting parallelism of the application by distributing the computations [4]. Our focus is to partition such that we can get the best runtime from the entire graph when executed sequentially. There are many challenges in designing such an intelligent partitioner in the context of DNNs:

- **Large search space.** Partitioning computation introduces inter-device data transfer, injecting dependency between operations and exploding the search space exponentially to the number of operations N. For a device with M compute devices, the search space becomes intractable - M^N. For example, for *ResNet-152*, with 160 operations and 2 devices, the search space is in the order of 10^{48}. The problem for general graphs is known to be *NP-complete* [4].
- **Varied DNN Topology.** DNN graphs can be of varied topology. Though an optimal setting can be inferred for a few type of graphs by designing specific algorithms for them, having a general purpose algorithm is challenging. For example, *AlexNet* can be represented as a linear Markov chain [10, 11], making it possible to apply a *Viterbi* style inference algorithm [7]. But, the same is not applicable to *ResNet* network.
- **Diverse Hardware Devices.** To support widely varying execution environments, a suitable partitioning system should be hardware agnostic, able to interact with the existing platform software, and to automatically characterize the device to find a suitable partitioning.

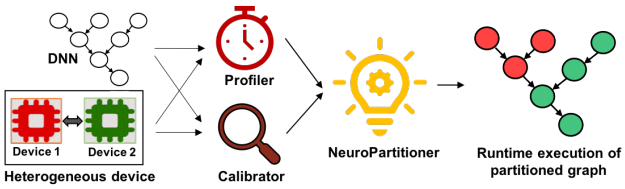

Figure 1. OverSeer: Automatically profiles, calibrates, partitions and executes a DNN on a heterogeneous device.

To this end, we introduce *OverSeer*, illustrated in Figure 1, that seamlessly integrates into platform software, automatically profiles the DNN application on heterogeneous devices, partitions, and executes the graphs on the computing resources. At the core of *OverSeer* is *NeuroPartitioner*, an algorithm that converts a DNN topology into a canonical format, finds suitable annotation and eventually partitions the graph. While partitioning, *OverSeer* employs *Profiler* to form an operator runtime database, while *Calibrator* uses a set of hardware-agnostic micro-benchmarks to automatically measure device parameters. The hardware-agnosticism is achieved by utilizing a deep learning compiler - TVM [1], enabling *OverSeer* to support varied execution environments.

The contributions of this paper are

- **Topology-Agnostic Partitioning.** *NeuroPartitioner*, a novel partitioning algorithm, that converts the DNN models into a canonical format, drastically reducing the search space and quickly estimating suitable partitions.
- **Hardware-Agnostic Infrastructure.** *OverSeer* utilizes hardware-agnostic micro-benchmarks, and uses TVM[1], to automatically measure hardware characteristics and support a wide-variety of execution environments.
- **Automatic End-to-end System.** *OverSeer automatically* profiles, partitions and executes the graphs to efficiently utilize the heterogeneous devices.

We observe that *OverSeer* is able to achieve up to 10% speedup for *ResNet-152* models on an *i7*-8700 HP machine with 6-Core CPU and an integrated GPU.

2 Design And Implementation

High-Level Design. *OverSeer* has three main components - *Profiler*, *Calibrator*, *NeuroPartitioner* as shown in Figure 1. *Profiler* generates an operator runtime database for each device. *Calibrator* uses a set of hardware-agnostic micro-benchmarks to automatically measure communication-related device characteristics like inter-device bandwidth and device-kernel invocation time. *NeuroPartitioner*, using the profiling and hardware measurements, converts the graph into a canonical format, and intelligently annotates it across the devices, which are then partitioned and executed via TVM. We now present the details of *NeuroPartitioner* algorithm.

Problem Definition. The partitioning problem can be formulated in a form of an annotated graph G where $G = (V, E, D, A, f, g)$ comprises of a set of vertices V representing DNN operations, a set of directed edges E denoting dependencies, a set of available devices D, a mapping annotation function $A = V \rightarrow D$ from vertices to devices, a cost function $f(v; \phi, A), v \in V$ to estimate the runtime of each op, and an inter-device communication cost function $g(e; \phi, A), e \in E$. ϕ is the database consisting of profiled computational and calibrated communication cost. Our objective is to minimize the computational and communication cost formulated as follows

$$\overset{*}{A} = \underset{A}{argmin} \sum_{v \in V} f(v; \phi, A)) + \sum_{e \in E} g(e; \phi, A) \quad (1)$$

NeuroPartitioner. The key challenge for the annotator is intractable search space. Relevant prior works have used iterative algorithms like *genetic algorithm* and *reinforcement learning*, but these can get stuck in local minima/maxima, possibly leading to highly sub-optimal results. Additionally, these iterative algorithms typically require considerable efforts to tune the accuracy.

We observe that typical DNNs graphs have some unique attributes that can help us reduce the search space. First, DNN graph G is typically a directed acyclic graph with a topological ordering and converges to a single sink (concatenating all outputs). Second, DNN graphs are typically static or can be easily reduced to static ones by breaking it down into completely static subgraphs. With these observations, an undirected loop can still exist if there is a node that has multiple children. But, this loop can also be broken by eliminating all-but-one child egdes. These characteristics lets us represent G in a specific tree format and significantly reduce the search space complexity. The process can be broken down into Canonicalizing and Device selection.

1. *Canonicalization*: First, we generate a spanning tree from G by following the path of the largest computation cost. This reduces G to a static converging polytree. This reduction allows us to design a heuristic that is polynomial to the number of nodes.
2. *Device Selection*: Then, we perform a top-down forward pass to find the best runtime for each node in the graph for each device, followed by a backtracking bottom-up phase selecting the device for each node that leads to the lowest runtime. This whole process can be easily represented as a dynamic programming problem whose cost value can be efficiently computed as follows,

$$cost_{v \in V, i \in D} = f(v; \phi, i, A_{v \rightarrow i}) + \underset{j \in D}{min}\{ \sum_{p \in P(v)} g(e_{p \rightarrow v}; \phi, A_{p \rightarrow j, v \rightarrow i}) + cost_{p,j}\} \quad (2)$$

Here $P(v)$ indicate the set of parent nodes of v. This significantly reduces the complexity to $O(|D|^2 \times |V|)$.

OverSeer takes this annotated graph, partitions it and executes the subgraphs to efficiently utilize the available compute resources.

3 Preliminary Results

We evaluate *OverSeer* using a variety of computer vision models on an i7-8700 HP machine, with 6-Core CPU and an integrated GPU. The findings of the experiment are shown in the Figure 2. We observe, that compared to a system that has no partitioning (best of all-GPU or all-CPU), *OverSeer* is capable of better utilizing the heterogeneous device, with a max of 10% for ResNet-152 network. This speedup includes the communication cost (data transfer and kernel invocation).

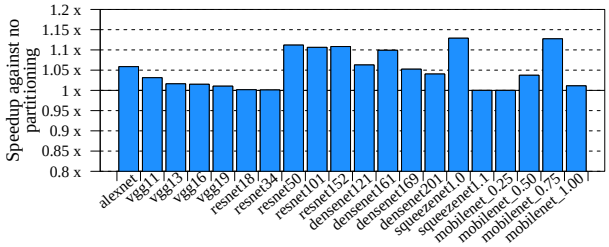

Figure 2. *OverSeer* achieves substantial speedup compared to no partitioning (best of all-CPU/all-GPU), efficiently utilizing heterogeneous compute capabilities.

4 Future Work

Though *OverSeer* observes a performance improvement on a heterogeneous device, our *NeuroPartitioner* algorithm is still a sub-optimal solution to the original NP-complete problem. Essentially, this is a search-space exploration vs optimal guarantee trade-off. An *all-GPU/CPU* has low guarantee as the search space is unexplored. A *greedy* allocation gives a higher guarantee with a better explored search-space whereas *genetic* algorithm explores all of the search space with a low guarantee. *OverSeer* explores more search space than *greedy* but less than *genetic* and comes with a better guarantee than both of them. Our future work is intended in further investigating algorithms that can explore more search space while holding similar or better guarantees.

Additionally, we will further solidify *OverSeer*'s hardware- and topology-agnosticism. We are currently working with multiple server and edge heterogeneous devices, yielding promising results for CV models. We plan to strengthen *NeuroPartitioner*'s topology-robustness by including new NLP networks in our evaluation set.

References

[1] Chen, T., Moreau, T., Jiang, Z., Zheng, L., Yan, E., Shen, H., Cowan, M., Wang, L., Hu, Y., Ceze, L., Guestrin, C., and Krishnamurthy, A. TVM: An automated end-to-end optimizing compiler for deep learning. In *13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18)* (Carlsbad, CA, Oct. 2018), USENIX Association, pp. 578–594.

[2] He, K., Zhang, X., Ren, S., and Sun, J. Deep residual learning for image recognition. *CoRR abs/1512.03385* (2015).

[3] He, K., Zhang, X., Ren, S., and Sun, J. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *2015 IEEE International Conference on Computer Vision (ICCV)* (Dec 2015), pp. 1026–1034.

[4] Kwok, Y.-K., and Ahmad, I. Static scheduling algorithms for allocating directed task graphs to multiprocessors. *ACM Comput. Surv. 31*, 4 (Dec. 1999), 406–471.

[5] Lane, N. D., Bhattacharya, S., Georgiev, P., Forlivesi, C., Jiao, L., Qendro, L., and Kawsar, F. Deepx: A software accelerator for low-power deep learning inference on mobile devices. In *2016 15th ACM/IEEE International Conference on Information Processing in Sensor Networks (IPSN)* (April 2016), pp. 1–12.

[6] LeCun, Y., Bengio, Y., and Hinton, G. Deep learning. *Nature 521*, 7553 (5 2015), 436–444.

[7] Rabiner, L. R., and Juang, B. H. An introduction to hidden markov models. *IEEE ASSp Magazine* (1986).

[8] Rotem, N., Fix, J., Abdulrasool, S., Deng, S., Dzhabarov, R., Hegeman, J., Levenstein, R., Maher, B., Satish, N., Olesen, J., Park, J., Rakhov, A., and Smelyanskiy, M. Glow: Graph lowering compiler techniques for neural networks. *CoRR abs/1805.00907* (2018).

[9] Salimans, T., Goodfellow, I. J., Zaremba, W., Cheung, V., Radford, A., and Chen, X. Improved techniques for training gans. *CoRR abs/1606.03498* (2016).

[10] Taskar, B., Guestrin, C., and Koller, D. Max-margin markov networks. In *Proceedings of the 16th International Conference on Neural Information Processing Systems* (2003), NIPS'03, pp. 25–32.

[11] Yedidia, J. S., Freeman, W. T., and Weiss, Y. Generalized belief propagation. In *Proceedings of the 13th International Conference on Neural Information Processing Systems* (2000), NIPS'00, pp. 668–674.