CoIn: Accelerated CNN Co-Inference through data partitioning on heterogeneous devices

Vanishree K*, Anu George*, Srivatsav Gunisetty†, Srinivasan Subramanian*, Shravan Kashyap R* and Madhura Purnaprajna*

*Dept of Computer Science & Engineering, Amrita School of Engineering, Bengaluru

†Dept of Computer Science & Engineering, Amrita School of Engineering, Amritapuri

Amrita Vishwa Vidyapeetham, India

(k vanishree,g anu,p madhura)@blr.amrita.edu,

srivatsav1998@gmail.com, srinivasansubramaniam74@gmail.com, shravanr97@outlook.com

***Abstract*—In Convolutional Neural Networks (CNN), the need for low inference time per batch is crucial for real-time applications. To improve the inference time, we present a method (CoIn) that benefits from the use of multiple devices that execute simultaneously. Our method achieves the goal of low inference time by partitioning images of a batch on diverse micro-architectures. The strategy for partitioning is based on offline profiling on the target devices. We have validated our partitioning technique on CPUs, GPUs and FPGAs that include memory-constrained devices in which case, a re-partitioning technique is applied. An average speedup of 1.39x and 1.5x is seen with CPU-GPU and CPU-GPU-FPGA co-execution respectively. In comparison with the approach of the state-of-the-art, CoIn has an average speedup of 1.62x across all networks.**

***Index Terms*—CNN, CPU-GPU system, GPU-FPGA system, Data Partitioning, inference acceleration**

I. Introduction

A CNN finds its use in applications in the domains of automotives, Internet of Things, search engines, medical appliances, emotion and speech recognition, image classification etc. [1]. For real-time applications, it is crucial to accelerate inference time. Existing inference acceleration systems focus on accelerating the algorithm [2]. Our model, CoIn (**Co**-execution of devices to accelerate CNN **In**ference) focuses on achieving inference acceleration by leveraging the available resources in CPU-GPU or CPU-GPU-FPGA systems. Devices in such systems have different individual performances for any given application because of their diverse micro-architectures. GPUs are good with data-parallel applications but with irregular programs, their performance gets degraded because of their SIMD architecture [3], [4]. Irregular applications perform better on CPUs and FPGAs than they do on GPUs [5]. But a GPU outperforms a CPU in a data-parallel application. Our exploration of CPU-GPU co-execution for MobileNet has shown an execution time improvement of 54.2% over the device with the least execution time, as observed in Figure 2. It has been our motivation to explore the use of devices in a CPU-GPU system for inference acceleration using data partitioning since it does not require an algorithmic modification effort. Similarly, inference acceleration achievable for CifarNet and SqueezeNet through CoIn has been explored for CPU-GPU-FPGA systems.

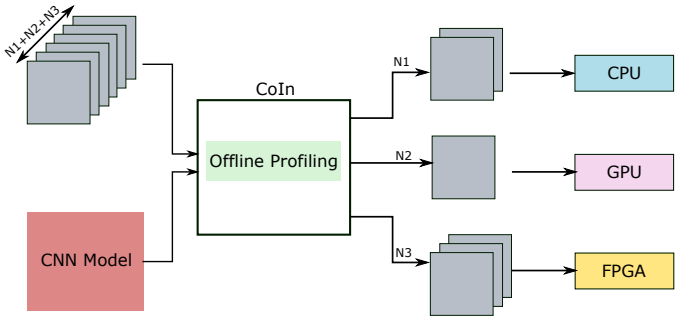

Fig. 1: CNN Inference Acceleration through data partitioning

As shown in Figure 1, CoIn estimates data partitioning ratio in terms of the number of images to be executed on each of the devices in a CPU-GPU-FPGA system to improve the inference time of a network. The approach of CoIn could be applied for real time applications such as driverless cars, ADAS wherein images are streamed from different cameras. Batches of these images can be processed in parallel among all the devices faster than on a single device.

In this work, our contributions through CoIn are as follows:

- Estimating relative performance of CNN models on a diverse set of CPUs, GPUs and FPGAs through offline profiling, for a finite set of batch sizes.
- Based on the relative performance, build a model to estimate data distribution ratio on the participating devices (CPU-GPU / CPU-GPU-FPGA co-inference systems) for any batch size.
- For memory constrained devices, estimate the data re-partitioning ratio. Further, determine the combination of CPUs, GPUs and FPGAs in the system that would improve inference time.

The rest of the paper is organized as follows. Section II describes the methodology of offline network characterization and subsequently, the methodology of estimating the data partitioning ratio among target devices. Section III and Section IV present the experimental setup and the results of our analysis. Section V discusses existing systems that cater to the challenge of accelerating training and inference phases and

978-1-7281-5197-7/20/$31.00 ©2020 IEEE

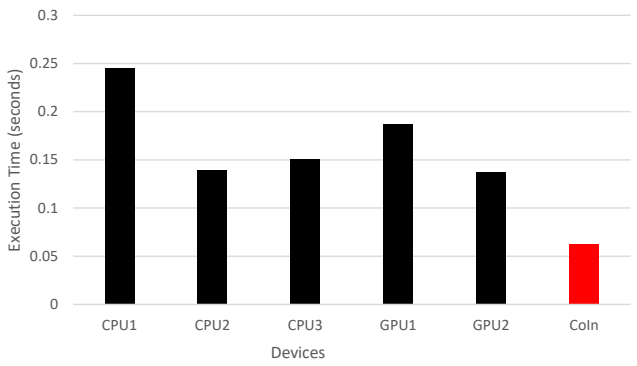

Fig. 2: CoIn outperforms other devices w.r.t inference time.

explains the uniqueness of our approach and analysis. Finally, Section VI draws conclusions based on our analysis performed in Section IV and briefs the possible future scope of our work.

II. Methodology

The goal of CoIn is to achieve high throughput inference of CNN models, while maintaining a given low latency (<7ms [6]). The approach of CoIn satisfies throughput requirements and avoids memory, latency penalty [7] by having batch sizes ranging from 2 to 100. For batch size=1 or 2, our model allocates the batch entirely to the fastest device on the platform. The computation on a single image is not distributed among the devices but the computation on the whole batch of images are distributed by employing model parallelism. The partition of batch sizes is static and hence, there is no runtime overhead involved in this approach. This work aims at improving the inference time with distribution of images. The accuracy of a network remains unchanged as its weights remain unchanged during inference. Given a batch size of images (>1), estimating the number of input images of a batch to be fed to each device based on their relative performance is data distribution.

CoIn achieves data partitioning by first characterizing a network for each of the devices by offline profiling. Based on the relative performance of the devices, it estimates the number of images of the batch to be assigned to each device. Next, it chooses appropriate devices as described in Figure 3. If the assigned number of images of the batch per device does not fit into a memory constrained device, then the images are redistributed as explained in the following subsections. Inference time for a batch of images when two or more devices execute simultaneously is termed as CoIn time, throughout this paper. The time taken for running a batch on a network deployed on a single device is termed as execution time. The performance metric used to validate CoIn is the CoIn time.

We construct Roofline based on Intel Haswell [8]CPU (CPU3) and Nvidia Maxwell GPU (GPU1) architecture, respectively as shown in Figures 4 and 5. For CPU, we use Intel Advisor to construct the roofline. For GPU, we construct based on a methodology [9] for Nvidia Maxwell GPUs. The x-axis denotes arithmetic intensity (flop/byte) and y-axis represents throughput (GFLOP/sec). These plots show the performances of MobileNet, GoogleNet, ResNet18 and SqueezeNet for different batch sizes. As seen in these plots, all the networks are compute-bound. An algorithm is compute bound when its arithmetic intensity (flop/byte) is greater than the ratio of peak FLOPs to peak bandwidth of the target device. The performance of the algorithm is closer or farther from peak FLOPs based on the capability of the device micro-architecture for which the roofline is constructed. CNNs are compute bound because of the significant compute intensive operations in the convolutional layers which dominate over the computation of the rest of the layers of the CNN model. The compute-bound property (higher flop/byte ratio) of the CNNs gives the utility of CoIn to achieve the best inference time through CPU-GPU / CPU-GPU-FPGA co-execution. These roofline plots are representative of the compute-bound nature for these networks on other devices.

We observe through offline profiling that execution time for a batch increases linearly on each device, for all networks. Therefore, the execution time of a network on a device is profiled offline only at two extreme batch sizes i.e. for the smallest and largest sizes. This linearity characterizes each network on each device by a linear equation and approximates execution time with an average error of 3.53%. Hence, we estimate execution time at any batch size using this linear equation.

Next, we estimate the data distribution ratio based on relative performance of the devices as given by Equation 1.

$$r_i = \frac{\frac{max(t)}{t_i}}{\sum_{j=1}^{n} \frac{max(t)}{t_j}} \quad (1)$$

where r_i is the data partitioning ratio for the device i, $max(t)$ is the maximum of execution times of all devices of the system, n is the number of devices in the system. Based on this partitioning ratio and the given batch size, the number of images of the batch assigned to each device is estimated. The better the performance of a network on a device, the larger the number of images assigned to that device. If the estimated partitioning ratio for CPU1, GPU1, GPU2 is 10%, 60% and 30% respectively, then CPU1, GPU1, GPU2 will receive 10, 60 and 30 images respectively when a batch contains 100 images. CoIn, then ensures the number of images assigned to each device meets that device's memory constraints. If it meets the constraints, then the ratio remains unchanged and the images are distributed as per the estimated ratio. If it does not meet the constraints, then CoIn modifies this ratio to re-balance the number of images accordingly. Re-balancing is the process of re-partitioning the remaining images (that do not fit in a device's memory) among the rest of the devices in the combination, according to their relative performances. CoIn time is the time taken by the slowest device in a chosen combination of devices.

Finally, the inference from each device is collected at the host. The time taken for data transfer through PCI from the

978-1-7281-5197-7/20/$31.00 ©2020 IEEE

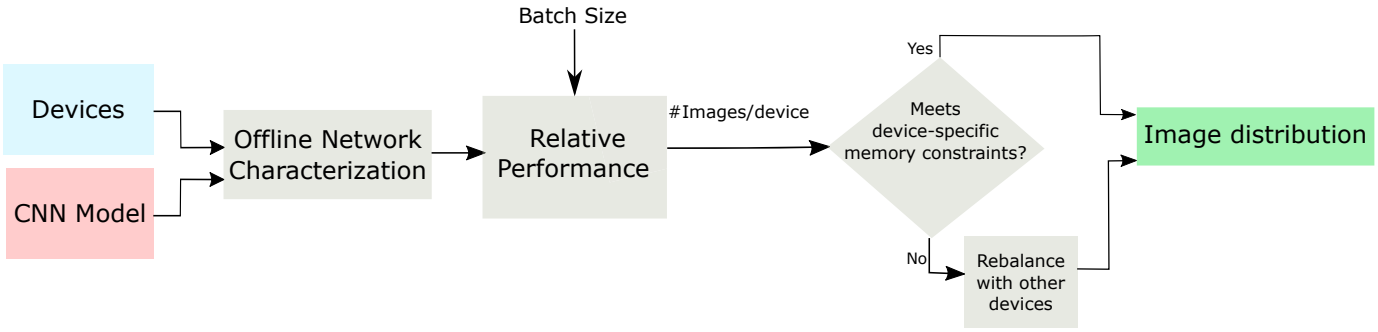

Fig. 3: CoIn Flow

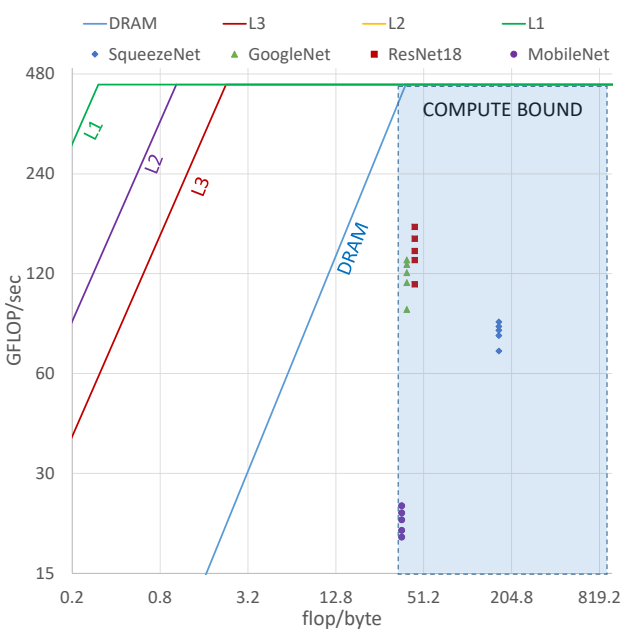

Fig. 4: Roofline model of CPU3 for different networks and batch sizes.

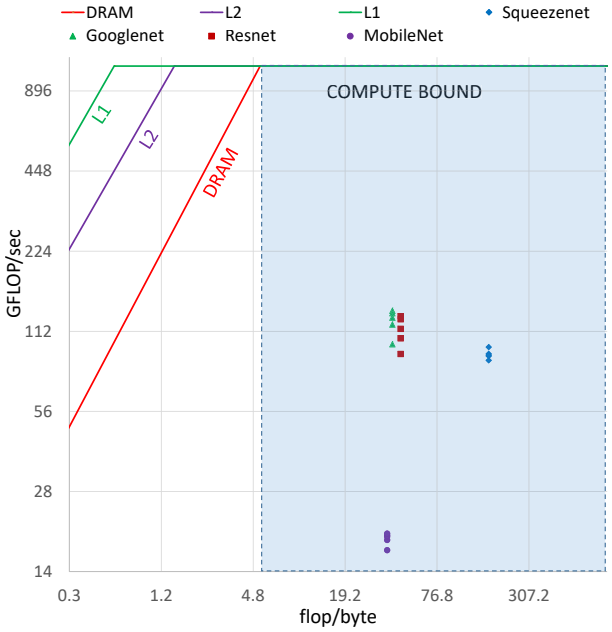

Fig. 5: Roofline model of GPU1 for different networks and batch sizes.

host to the devices, and inference back to the host is taken into account in the estimation of execution time of the GPUs and FPGA in the system. This ensures that the data partitioning ratio is estimated with all the latencies accounted for in the execution of the end-to-end inference phase of the network.

III. Experimental Setup

The devices used in our experiments to evaluate our model are listed in Table I. We measure and estimate GPU and FPGA execution time that includes the time for data transfer between devices and the host through PCI. We have limited ourselves to systems that have 1 CPU and 1 or more GPUs or FPGA, because systems with 2 CPUs communicate through MPI [10],which involves, over the cluster, additional latency, that has to be taken into account in the inference time estimation, which extends beyond our existing system setup.

We use the Caffe [11] framework that is optimized for Intel CPUs and Nvidia GPUs. The compiler used for the CPU and GPU flows is *gcc* version 4.8.4. For CUDA compilation, *nvcc* version 8.0 is used and for measuring execution time and power, *nvprof* is used. For the CPUs, energy is measured using Intel's PCM tool. CPU power is obtained by dividing the measured energy by the measured execution time. Details of the networks, the data sets and the maximum batch sizes are in Table II.

Intel's OpenVino Toolkit [12] is used along with Caffe on CPU3; Tango [13] framework is used on GPU1, FPGA for CPU3-GPU1-FPGA. For the FPGA, Xilinx Vivado Design Suite (2018.3) is used for the compilation and measurement of execution time and power.

IV. Results and Evaluation

We evaluate CoIn time estimation error and compute the performance improvement obtained for the chosen devices.

978-1-7281-5197-7/20/$31.00 ©2020 IEEE

TABLE I: Architectural specification of devices used

Device	Specifications
CPU1	Intel Core i7-7500U @2.70GHz (2 physical cores, 4 logical cores)
CPU2	Intel Core i7-6700HQ @2.60GHz (4 physical cores, 8 logical threads)
CPU3	Intel(R) Core(TM) i7-4770 CPU@3.4GHz (4 physical cores, 8 logical threads)
GPU1	Nvidia GPU GTX750 @1020MHz (512 CUDA cores), 1GB memory
GPU2	Nvidia GPU GTX TITAN Black @889GHz (2880 CUDA cores), 4GB memory
FPGA	Xilinx Virtex7 xc7v200tfhg1761
PCIe BW	4GB/s (PCI Ver 1.0, width x16)

TABLE II: Description of CNNs evaluated. Execution time is measured for the batch sizes that are limited by a memory constrained device.

Network	Dataset	Batch Size
Cifar10Net	CIFAR-10	100
LeNet	MNIST	100
SqueezeNet	ImageNet	18
MobileNet	ImageNet	6
GoogleNet	ImageNet	15
ResNet-18	ImageNet	10

A. Performance improvement

The memory constraints of the chosen devices affect the performance improvement obtained with co-execution.

1) Batch size within memory constraints: These experiments are carried out for batch sizes that meet the memory constraint of the lowest memory device. Figure 6 shows the plot for speedup of CoIn w.r.t. CPUs and GPUs used in our experiments. An average CoIn performance speedup of 1.39x is seen over all devices across all networks. It is also observed that CoIn performance is best when all devices execute simultaneously. Hence, this demonstrates that the usage of all devices in a node, improves inference time.

2) Batch size exceeds memory constraints: We also evaluate CoIn with a bigger batch size for a memory-constrained device. GPU1 has the lowest memory among all the devices used in our experiments. In the case of MobileNet, only 6 images can fit in GPU1's memory. If a batch size of 100 is specified, then, we try to determine whether the 3 chosen devices perform better than the combination of 2 devices or a single device.

In this case, the estimated partitioning ratio for the chosen devices, CPU2-GPU1-GPU2=36:27:37 is modified such that GPU1 is assigned only 6 images. The remaining 21 images are distributed between CPU2 and GPU2 based on their estimated partitioning ratio, which is 50:50. Hence, the number of images input to CPU2 and the number of images input to GPU2 increase to 46 and 48 respectively. As shown in Figure 7, the CoIn time with CPU2-GPU1-GPU2 outperforms other combinations with this re-balancing approach for MobileNet. CoIn exhibits the same behaviour for other networks in the plot. Performance improvement achieved, by re-balancing, across the four networks, is 11.16%.

3) Performance improvement on CPU-GPU-FPGA platform: CifarNet and SqueezeNet architectures are evaluated on CPU3-GPU1-FPGA. Performance improvement of 43.88% and 56.89% is obtained for CifarNet and SqueezeNet respectively, with data partitioning. Speedup of CoIn time with data partitioning is shown in the Figure 8.

Energy consumption is calculated for CifarNet and SqueezeNet. It is observed that with data partitioning, the energy consumption for SqueezeNet is 15% lower than the energy consumption for SqueezeNet on the fastest device (GPU1).

B. Comparison with the state-of-the-art

We compare CoIn with CcT [2], the only state-of-the-art framework that performs work scheduling on CPUs and GPUs based on peak TFLOP/sec of devices. It modifies the measured relative performance of the CPU and GPU in a node so as to bring it closer to their peak relative performance(TFLOP/sec). It has been tested on the GPUs and CPUs available on Amazon EC2 instance. Hence, their approach can bring optimal partitioning only on those CPUs and GPUs. In contrast, CoIn can work on any GPU or CPU since it is based on offline profiling. Table III shows the speedup of CoIn w.r.t. inference time obtained by CcT partitioning technique for the CPU and GPU devices used in our experiments. An average speedup of 1.62x is observed over the CcT approach.

TABLE III: Comparison with CcT for a batch size of 100 for one iteration for all networks. This comparison is performed on the CoIn time with CPU2-GPU1-GPU2. For MobileNet, CcT's partitioning ratio allots more than 6 images on GPU1 and hence, memory is insufficient.

Network	CcT(secs)	CoIn(secs)	Speedup
Cifar10Net	0.017	0.0076	2.23
LeNet	0.0087	0.0048	1.78
SqueezeNet	0.2842	0.21	1.34
MobileNet	Memory insufficient	0.94	-
GoogleNet	0.6062	0.416	1.45
resnet-18	0.5041	0.38	1.318

V. Related Work

Different workload partitioning strategies are used for accelerating CNN training/inference on heterogeneous platforms. When the sizes of training data and model increase, it becomes impossible to perform training/inference on a single device [3]. The highly data-parallel nature of the computations such as convolution in CNN makes acceleration possible on distributed platforms [14]. It has resulted in extensive research for decreasing the training/inference time on distributed platforms by using data parallelism [2], [15] and model parallelism [14], [16], [17]. Data partitioning for data parallel models is mostly based on profiling [16], [17] and peak device performance [2], [15]. Data-parallelism is preferred to model-parallelism as the amount of communication overhead incurred is less with data parallelism.

978-1-7281-5197-7/20/$31.00 ©2020 IEEE

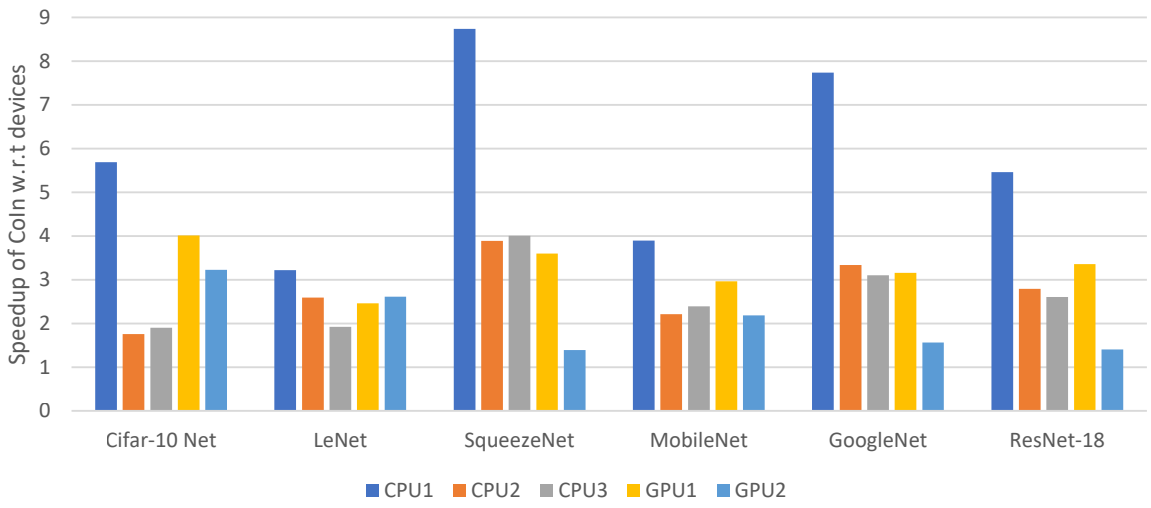

Fig. 6: Speedup of CoIn w.r.t all CPUs and GPUs.

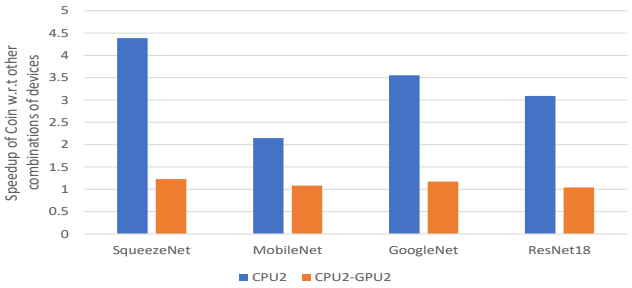

Fig. 7: Speedup of CoIn obtained by re-balancing strategy for a larger batch size that is limited by a memory-constrained device. Here, GPU1 has the least memory.

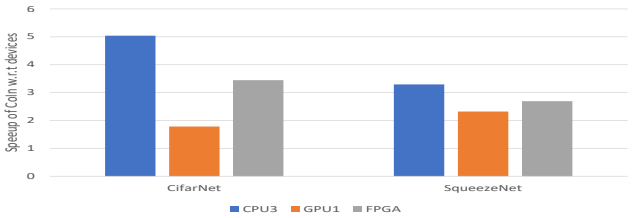

Fig. 8: Speedup of CoIn w.r.t. devices CPU3, GPU1, FPGA.

In our approach, data parallelism is used for CNN inference. This approach is similar to the strategy used by Hadjis et al. [15] for accelerating CNN training on CPU-only, GPU-only and CPU+GPU clusters. But, unlike the data partitioning strategy used by Hadjis et al., we use relative performance ratio corresponding to different devices and accelerate end-to-end network, as opposed to, only convolution layer. This ratio is statically determined based on the performance of each device in the platform for a batch size. Inference acceleration is achieved by choosing the optimum data partition for each device. Partitioning the batch to multiple devices, implemented in our work, also helps to run bigger models on memory constrained devices [14].

VI. Conclusions and Future Work

We have developed a co-execution inference model, CoIn, that accelerates CNN inference through co-execution among CPU, GPU and FPGA in a CPU-GPU-FPGA system. To achieve this, CoIn estimates the required data partitioning ratio based on the relative performance of the devices. It distributes data among a chosen combination of devices in a system, while meeting their memory constraints.

With co-execution among the chosen devices, we observe an average speedup of 1.39x and 1.5x with CPU-GPU and CPU-GPU-FPGA co-execution, respectively, for a batch across all networks. This method is generic and can be applied to any kind of device that can be characterized to different networks.

Our future work involves extending our methodology to clusters, other heterogeneous devices, many-core architectures and embedded platforms.

Acknowledgment

This research was conducted within the Indo-Swedish joint project titled ”Towards Next Generation Embedded Systems: Utilizing Parallelism and Reconfigurability”, funded by the Department of Science and Technology (INT/SWD/VINN/p-10/2015), Government of India and the Visvesvaraya Fellowship from Ministry of Electronics and IT, India.

References

[1] Suchitra, P. Suja, and S. Tripathi, “Real-time emotion recognition from facial images using Raspberry Pi ii,” in *Signal Processing and Integrated Networks (SPIN), 2016 3rd International Conference on*. IEEE, 2016, pp. 666–670.

[2] S. Hadjis, F. Abuzaid, C. Zhang, and C. Ré, “Caffe con troll: Shallow ideas to speed up deep learning,” in *Proceedings of the Fourth Workshop on Data analytics in the Cloud*. ACM, 2015, p. 2.

[3] S. Lee, D. Jha, A. Agrawal, A. Choudhary, and W.-k. Liao, "Parallel deep convolutional neural network training by exploiting the overlapping of computation and communication," in *High Performance Computing (HiPC), 2017 IEEE 24th International Conference on*. IEEE, 2017, pp. 183–192.

[4] K. Vanishree and M. Purnaprajna, "Work in progress: Performance modeling for data distribution in heterogeneous computing systems," in *2018 International Conference on Compilers, Architectures and Synthesis for Embedded Systems (CASES)*. IEEE, 2018, pp. 1–3.

[5] M. Burtscher, R. Nasre, and K. Pingali, "A quantitative study of irregular programs on gpus," in *2012 IEEE International Symposium on Workload Characterization (IISWC)*. IEEE, 2012, pp. 141–151.

[6] N. P. Jouppi, C. Young, N. Patil, D. Patterson, G. Agrawal, R. Bajwa, S. Bates, S. Bhatia, N. Boden, A. Borchers *et al.*, "In-datacenter performance analysis of a tensor processing unit," in *Proceedings of the 44th Annual International Symposium on Computer Architecture*. ACM, 2017, pp. 1–12.

[7] "NVIDIA AI Inference Platform," https://www.advancedhpc.com/wp-content/uploads/2018/10/NVIDIA-AI-INFERENCE-PLATFORM.pdf, accessed: 2018.

[8] P. Hammarlund, A. J. Martinez, A. A. Bajwa, D. L. Hill, E. Hallnor, H. Jiang, M. Dixon, M. Derr, M. Hunsaker, R. Kumar *et al.*, "Haswell: The fourth-generation intel core processor," *IEEE Micro*, vol. 34, no. 2, pp. 6–20, 2014.

[9] A. Lopes, F. Pratas, L. Sousa, and A. Ilic, "Exploring gpu performance, power and energy-efficiency bounds with cache-aware roofline modeling," in *2017 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*. IEEE, 2017, pp. 259–268.

[10] W. Gropp, R. Thakur, and E. Lusk, *Using MPI-2: Advanced features of the message passing interface*. MIT press, 1999.

[11] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell, "Caffe: Convolutional architecture for fast feature embedding," *arXiv preprint arXiv:1408.5093*, 2014.

[12] "OpenVINO Toolkit, Intel Corporation,," https://software.intel.com/en-us/openvino-toolkit., [Online; accessed 26-July-2019].

[13] A. Karki, C. P. Keshava, S. M. Shivakumar, J. Skow, G. M. Hegde, and H. Jeon, "Tango: A deep neural network benchmark suite for various accelerators," *arXiv preprint arXiv:1901.04987*, 2019.

[14] S.-X. Zou, C.-Y. Chen, J.-L. Wu, C.-N. Chou, C.-C. Tsao, K.-C. Tung, T.-W. Lin, C.-L. Sung, and E. Y. Chang, "Distributed training large-scale deep architectures," in *International Conference on Advanced Data Mining and Applications*. Springer, 2017, pp. 18–32.

[15] S. Hadjis, C. Zhang, I. Mitliagkas, D. Iter, and C. Ré, "Omnivore: An optimizer for multi-device deep learning on cpus and gpus," *arXiv preprint arXiv:1606.04487*, 2016.

[16] T. M. Chilimbi, Y. Suzue, J. Apacible, and K. Kalyanaraman, "Project adam: Building an efficient and scalable deep learning training system." in *OSDI*, vol. 14, 2014, pp. 571–582.

[17] J. Marques, G. Falcao, and L. A. Alexandre, "Distributed learning of cnns on heterogeneous cpu/gpu architectures," *arXiv preprint arXiv:1712.02546*, 2017.