Hierarchical Design Space Exploration for Distributed CNN Inference at the Edge

Xiaotian Guo[1,2], Andy D. Pimentel[1], and Todor Stefanov[2]

[1] University of Amsterdam, The Netherlands, {x.guo3, a.d.pimentel}@uva.nl
[2] Leiden University, The Netherlands, t.p.stefanov@liacs.leidenuniv.nl

Abstract. Convolutional Neural Network (CNN) models for modern applications are becoming increasingly deep and complex. Thus, the number of different CNN mapping possibilities when deploying a CNN model on multiple edge devices is vast. Design Space Exploration (DSE) methods are therefore essential to find a set of optimal CNN mappings subject to one or more design requirements. In this paper, we present an efficient DSE methodology to find (near-)optimal CNN mappings for distributed inference at the edge. To deal with the vast design space of different CNN mappings, we accelerate the searching process by proposing and utilizing a multi-stage hierarchical DSE approach together with a tailored Genetic Algorithm as the underlying search engine.

1 Introduction

Convolutional Neural Networks (CNNs) have been intensively researched and widely used in many domains, including audio recognition, computer vision, and natural language processing. Since CNNs became the state-of-the-art in large-scale visual recognition and classification, countless advancements in improving CNN models have been made to solve traditionally challenging problems such as image recognition, classification, etc. Deploying these modern CNN models and performing the inference directly on an edge device is typically not possible because of limited resources in terms of memory capacity, computation capacity, and power budget of the edge device. Therefore, to perform CNN inference on edge devices, users typically need to rely on additional compute resources provided as service by the cloud. Realizing CNN inference on edge devices using such cloud services requires users to communicate a substantial amount of data between an edge device and a cloud server. Such data communication may cause data privacy concerns as well as low device responsiveness due to data transmission delays or temporal unavailability of the cloud services.

One approach to address the above problems and achieve CNN inference on an edge device without cloud services, is to perform CNN model compression, such as pruning, quantization, or knowledge distillation[6] that will allow to deploy the entire CNN model on the device. However, such an approach sacrifices the accuracy of the model to some extent, especially when high model compression rates are required. Another approach is to deploy only a part of the CNN

model on the edge device and the rest of the model in the cloud. Such an approach [10], however, still suffers from data privacy and cloud communication latency concerns. A third approach, which solves the aforementioned issues of the other two approaches, is to partition the CNN model and distribute the partitions across multiple edge devices to collaboratively perform the CNN inference. A general direction is to utilize model and data parallelism methods [7] to divide CNN computations over a number of edge devices. Such distributed execution of the CNN model inference often needs to take multiple requirements into account, like latency, throughput, resource usage, power/energy consumption, etc. Here, the way how the different CNN layers are distributed and mapped onto the edge devices plays a key role in optimizing/satisfying these requirements. For example, using model-parallelism techniques and mapping CNN layers in a balanced way may reduce the maximum per-device memory footprint or energy consumption. Or, some CNN mappings may generate a balanced data processing pipeline, thereby improving the overall throughput. As CNN models for modern applications are becoming increasingly deep and complex, the number of different CNN mapping possibilities when deploying multiple edge devices, and the various compute resources in each of them, is vast. Efficient Design Space Exploration (DSE) methods are therefore essential to find a set of (near-)optimal CNN mappings subject to one or more design requirements (i.e., objectives).

In this paper, we present an efficient DSE methodology to find optimal CNN mappings for distributed inference at the edge. To this end, we leverage our AutoDiCE framework [1] to assess the quality (in terms of inference throughput, memory footprint, and energy consumption) of a particular CNN mapping. AutoDiCE is a fully automated framework for distributed CNN inference over multiple edge devices. To deal with the vast design space of different CNN mappings, we accelerate the searching process by using a multi-stage hierarchical DSE approach together with a tailored Genetic Algorithm (GA) as the underlying search engine. At every stage, we perform DSE at two hierarchical levels. In the first level, we use analytical models inside a GA to approximate each objective function (i.e., throughput, memory, and energy consumption) to avoid relatively long evaluation times through real on-device (i.e., on-board) measurements using our AutoDiCE framework. The near-optimal solutions found in the first level together with Pareto-optimal solutions from a previous DSE stage are utilized as the parents for the second level DSE. In this second level, we evaluate each design point using real measurements taken from AutoDiCE-generated CNN inference implementations to determine the Pareto front for a next DSE stage. The output of the last DSE stage provides the final Pareto-optimal solutions. Our contributions can be summarized as follows:

- enhance the DSE process by creating analytical models to approximate each objective function in order to reduce the on-board evaluation cost during the DSE process;
- accelerate the DSE convergence by performing the DSE process in multiple stages where, at each DSE stage, we consider only specific part of the design

space and use as input Pareto-optimal solutions from the previous DSE stage in order to find Pareto-optimal solutions for the next DSE stage;
– improve the searching efficiency with a tailored chromosome encoding method, thereby scaling down the search space.

2 Related work

Today's prevalent CNN models for computer vision tasks are becoming increasingly large. Their execution, i.e. model inference, requires increasing amounts of memory and compute resources, putting a large burden on the cloud infrastructure. Offloading parts of a single CNN model to the edge has gained the attention of researchers to relieve the pressure on the cloud. For example, Neurosurgeon [10] vertically partitions a CNN model between a single edge device and the cloud. DDNN [17] also tries to partition a model between the cloud and edge devices, but model retraining is needed for each early-exit branch. However, the methods in [10, 17] execute only the first few layers of a CNN model at the edge, after which the rest of the computation is still offloaded to the cloud. The unpredictable low responsiveness and data privacy issues are still present in such partitioned CNN inference due to the partial involvement of the cloud [5]. To perform CNN inference on a fully distributed system at the edge, without any cloud involvement, data partitioning or CNN model partitioning is often required. For example, in [19], a data partitioning strategy is used in an object detection CNN-based application to split input data frames. Alternatively, CNN model partitioning splits CNN layers and/or connections of a large CNN model, thereby creating several smaller sub-models (partitioned models) where each sub-model is executed on a different edge device [16]. For instance, Hadidi *et al.* [7] exploits model-partition methods to perform single-batch inference over several collaborative and resource-constrained edge devices and utilizes their aggregated computing power via a local network. In addition to using data and CNN model partitioning to map large CNNs on resource-constrained edge devices, researchers try to optimize the CNN mapping to improve the inference performance. For example, the methodologies in [20, 18, 9] propose efficient algorithms to determine partitioning policies that generate efficient CNN mappings in order to improve the performance of cooperative inference over multiple edge devices. However, these methodologies optimize and evaluate CNN mappings based on analytical models only and consider limited number of objectives. In contrast, our DSE methodology optimizes more objectives, and besides analytical models, it uses AutoDiCE to evaluate mappings by real on-device measurements.

Distributed inference of large CNN models typically needs to consider a range of different design requirements, such as latency, throughput, resource usage, power/energy consumption, etc. These requirements/objectives can be conflicting, implying that there usually does not exist a single optimal CNN mapping that satisfies all requirements. Typically multiple solutions, so-called Pareto optimal solutions, co-exist and the set of all optimal solutions is called the Pareto front. Finding these Pareto-optimal CNN mappings for a given number

of edge devices to perform distributed CNN inference under several requirements is the topic of study in this paper. A popular approach to perform such a search for Pareto-optimal solutions is by using multi-objective evolutionary algorithms [4]. More specifically, in the domain of DSE, multi-objective Genetic Algorithms (GAs), such as the Non-dominated Sorting Genetic Algorithm (NSGA-II) [3], are widely used and have demonstrated to produce good results [15]. For instance, [11, 12] use the NSGA-II GA to explore the design space to find improved neural network architectures for CNN-based applications. Our DSE methodology also employs NSGA-II to explore the Pareto-optimal CNN mapping solutions with respect to throughput, maximum memory usage per device, and maximum energy consumption per device. However, NSGA-II can easily get stuck in so-called dominance resistant solutions [14], that are far away from the true Pareto front. How to search the optimal CNN mappings for distributed inference using NSGA-II, and efficiently find the Pareto front in the huge search space, are the main challenges we try to tackle in this paper.

3 Evaluation Methods

In this section, we discuss two different methods to evaluate the three objectives at every stage in our two-level DSE. The first level DSE applies analytical models to approximate the objectives, and the second level uses our AutoDiCE framework to evaluate the objectives of distributed CNN inference by real implementations and measurements on hardware boards.

3.1 Analytical Models

In the first level, we adopt analytical models to approximate the system throughput, memory usage, and energy consumption for each CNN mapping. We use t_{l_j}, M_{l_j}, E_{l_j} to represent the execution time, the memory usage, and the energy consumption of layer l_j in a CNN model, respectively. A CNN mapping $\mathbf{x}$ is denoted as $\mathbf{x} = [x_1, x_2, \cdots, x_L]$, where L is the number of layers in the CNN model and $x_j = PE_i$ means that layer l_j is mapped on processing element PE_i, which could, e.g., be a CPU or GPU inside an edge device. For a given mapping $\mathbf{x}$, the three objectives of the distributed system can be computed as follows.

Throughput The overall system throughput T_{system} is defined as the images processed per second (IPS) over multiple PEs:

$$T_{system} = \frac{1}{\max_{1 \le i \le N} (t^i)}; \qquad t^i = \sum_{\forall j : 1 \le j \le L \wedge x_j = PE_i} t_{l_j} + t_{comm}$$

where t^i is the time to process one image on PE_i, N is the total number of deployed PEs in the distributed system, and t_{comm} is the time needed for data communication related to PE_i. We assume that the size of input images are already determined as well as the input and output tensors of every CNN layer are also fixed. Then, we can estimate the total number of operations in every

layer and the total size of communicated data related to PE_i. The execution time t_{l_j} is estimated through the number of multiply–accumulate operations (MACs). A proper approximation for communication time t_{comm} depends on data movements, and involves intra-node shared memory communication, intra-node communication between CPU and GPU, or inter-node communication over the network.

Memory Every PE_i allocates memory M^i which consists of three parts: memory for CNN coefficients (i.e. weights, bias, and parameters), memory for output buffers to store intermediate results of layers, and memory for input buffers of some layers to receive data from other PEs:

$$M^i = \sum_{\forall j:1\leq j\leq L\wedge x_j=PE_i} (M^j_{coeffs} + M^j_{outbuffs} + M^j_{inbuffs})$$

where M^j_{coeffs}, $M^j_{outbuffs}$, and $M^j_{inbuffs}$ denote the sizes of the aforementioned memory parts associated with layer l_j mapped on PE_i. These sizes (in number of elements) are approximated based on the type of CNN layer l_j. For example, given a convolutional layer l_j, the memory sizes are calculated as follows:

$$M^j_{coeffs} = w_k * h_k * C_{in} * C_{out} + C_{out}$$

$$M^j_{outbuffs} = w_{out} * h_{out} * C_{out} \qquad M^j_{inbuffs} = w_{in} * h_{in} * C_{in}$$

where w_k and h_k are the width and height of the convolution kernel, C_{in} and C_{out} are the number of input and output channels of layer l_j, and w_{in}, h_{in}, w_{out}, h_{out} are the width and height of the input and output tensors of layer l_j. If layer l_j mapped on PE_i does not receive data from layers that are mapped on other PEs then $M^j_{inbuffs} = 0$.

Energy Every PE_i consumes energy E^i to execute the CNN layers mapped on PE_i. In our energy consumption analytical model, E^i includes the energy consumed for inference computation and data communication with other PEs:

$$E^i = \sum_{\forall j:1\leq j\leq L\wedge x_j=PE_i} E^j_{comp} + \sum_{\forall j:1\leq j\leq L\wedge x_j=PE_i} E^j_{comm}$$

where E^j_{comp} and E^j_{comm} denote the computation and communication energy consumption for layer l_j, respectively. Here, E^j_{comm} is non-zero only when layer l_j actually communicates with another PE. We calculate E^j_{comp} and E^j_{comm} as follows:

$$E^j_{comp} = \int_0^{t^j_{comp}} P^j_{comp}(t)\, dt; \qquad E^j_{comm} = \int_0^{t^j_{comm}} P^j_{comm}(t)\, dt$$

where $P^j_{comp}(t)$ is the power consumption during the execution of layer l_j, and $P^j_{comm}(t)$ is the power consumption during data communication of layer l_j with another PE. $P^j_{comp}(t)$ and $P^j_{comm}(t)$ can be acquired by measurements during CNN layer profiling on an edge device.

3.2 AutoDiCE Framework

As explained in Section 1, in the second level we use our AutoDiCE framework [1] to evaluate the fitness (i.e., the quality) of a given CNN mapping in terms of the following objectives: CNN inference throughput, maximum memory usage per device, and maximum energy consumption per device. AutoDiCE enables us to evaluate the quality of a CNN mapping through actual on-device measurements. AutoDiCE is a fully automated framework for distributed CNN inference over multiple edge devices. Given a specific input CNN model and specifications of the model partitioning and mapping of the partitions to (various resources within) multiple edge devices, AutoDiCE automates the actual model partitioning, code generation, and deployment of the CNN partitions on the edge devices.

Figure 1 shows the user interface and design flow of AutoDiCE, where the main steps in the flow are divided into two modules: front-end and back-end. The interface is composed of three specifications, namely a Pre-trained CNN Model (provided as an .onnx file), Mapping Specification (a .json file), and Platform Specification (a .txt file). The Pre-trained CNN Model specification includes the CNN topology description with all layers and connections among layers as well as the weights/biases that are associated with the layers and obtained by training on a specific dataset using deep learning frameworks like PyTorch, TensorFlow, etc. Many such CNN model specifications in ONNX format [2] are readily available in open-access libraries and can be directly used as an input to the framework. The Platform Specification lists all available edge devices together with their computational hardware resources and specific software libraries associated with these resources. The Mapping Specification is a list of key-value pairs that explicitly specifies how all layers described in the Pre-trained CNN Model specification are mapped onto the computational hardware resources listed in the Platform Specification. Every unique key corresponds to an edge device with a selection of its hardware resources, like CPUs or GPU, to be used for computation. Every value corresponds to a set of CNN layers to be deployed and executed on the edge device resources. Such Mapping Specification can be provided manually by the user or, like in this paper, generated by an external mapping DSE tool.

The three aforementioned specifications are given as an input to the front-end module as shown in Figure 1, which then performs two main steps: *Model Splitting* and *Config & Communication Generation*. The Model Splitting takes as an input the Pre-trained CNN Model and Mapping specifications, splits the input CNN model into multiple sub-models, and generates these sub-models in ONNX format. The number of generated sub-models is equal to the number of unique key-value pairs in the Mapping Specification. The Config & Communication Generation step takes all three input specification files and generates specific tables in JSON format containing information needed to realize proper communication and synchronization among the sub-models using the well-known MPI interface. In addition, a configuration text file (MPI rankfile) is generated to initialize and run the sub-models as different MPI processes.

The back-end module subsequently uses the output from the front-end for code and deployment package generation. During the *Code Generation* step, effi-

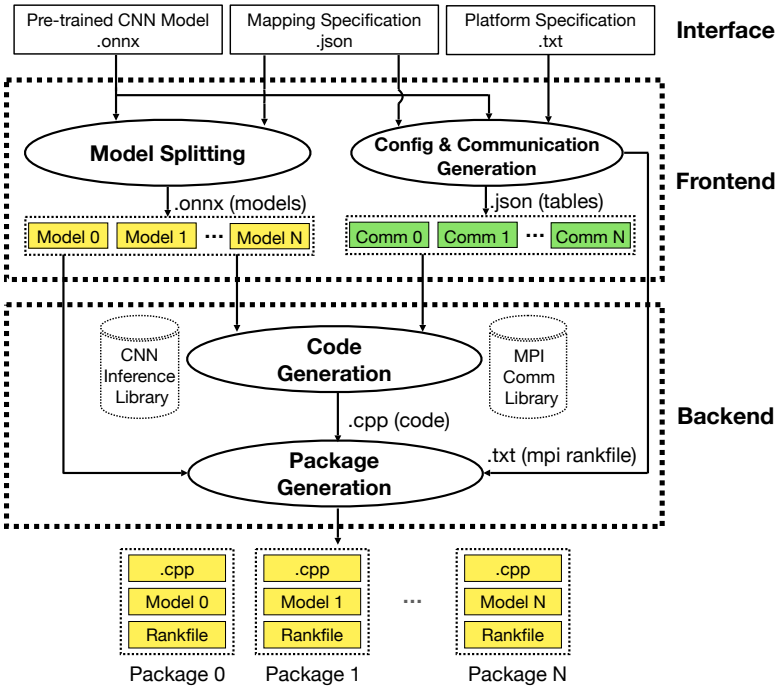

Fig. 1. The AutoDiCE design flow and its user interface

cient C++ code is generated for every edge device based on the input sub-models and tables. In the generated code, primitives from the standard MPI library are used for data communication and synchronization among sub-models as well as primitives from a custom CNN Inference Library are used for implementation of the CNN layers belonging to every sub-model. This CNN Inference Library also integrates OpenMP support. This means that if a CNN layer is mapped onto multiple CPU cores in an edge device, the actual execution of such layer will be multi-threaded using OpenMP to efficiently utilize the multiple CPU cores by exploiting data parallelism available within the layer. Finally, the *Package Generation* step packs the generated C++ code, the MPI rankfile, and a sub-model together to generate a specific deployment package for every edge device.

4 Multi-stage Hierarchical Design Space Exploration

Our DSE methodology utilizes a Genetic Algorithm (GA), namely the NSGA-II algorithm [3], to search for optimal mappings of (complete) CNN layers to different, distributed edge devices. We assume that each edge device contains a number of internal compute resources (i.e. PEs), like a CPU and GPU, and we map CNN layers directly to these specific PEs within an edge device.

Given a trained CNN model with L layers, a layer l_j performs a computation operation in the CNN model such as a convolution (Conv), a matrix

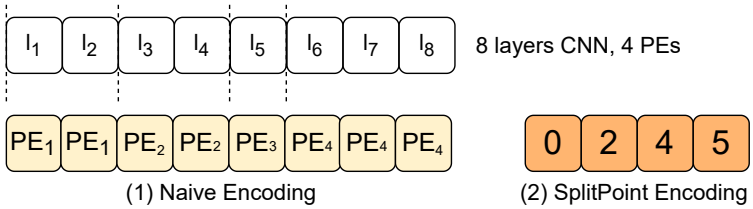

Fig. 2. Two Chromosome Encoding Methods

multiplication (FC), etc. As mentioned in Section 3.1, a mapping $\mathbf{x}$ of the CNN layers onto a total of N PEs is denoted as $\mathbf{x} = [x_1, x_2, \cdots, x_L]$. Such mapping notation $\mathbf{x}$ is typically encoded with the GA's chromosome where $PE_i, i \in [1..N]$ define the gene types in the chromosome. An example of such encoding, called Naive Encoding (NE), is shown in Figure 2. The GA chromosome [PE_1,PE_1,PE_2,PE_2,PE_3,PE_4,PE_4,PE_4] encodes an 8-layer CNN ($L = 8$) mapped onto four PEs ($N = 4$), where layers l_1 and l_2 are mapped on PE_1, l_3 and l_4 on PE_2, l_5 on PE_3, and l_6, l_7, l_8 on PE_4. Such naive encoding for CNN mappings is simple and intuitive but it may require exploration of a huge design space because the space size depends exponentially on the number of layers L in a CNN model and L is typically large. Therefore, in our DSE methodology, we propose and utilize a tailored chromosome encoding method, called Split Point Encoding (SPE). It encodes points in a CNN model that partition the model into N groups of CNN layers, where each group consists of consecutive layers and is mapped on one PE. In Figure 2, the Split Point Encoding example encodes the same mapping as the Naive Encoding example. It can be seen that the 8-layer CNN has four split points, visualized with the vertical dashed lines, at positions 0, 2, 4, and 5 determined by the layer index j. Therefore, the GA chromosome using our SPE method is [0, 2, 4, 5] and it encodes four groups of layers each mapped on one PE as follows: 1) for $j \in (0..2]$, l_j mapped on PE_1; 2) for $j \in (2..4]$, l_j mapped on PE_2; 3) for $j \in (4..5]$, l_j mapped on PE_3; 4) for $j > 5$, l_j mapped on PE_4. The length of our SPE chromosome is equal to the number of PEs which is N, thus SPE requires exploration of a design space which size depends exponentially on N. Since N is typically much smaller than the number of CNN layers L, our SPE method largely scales down the design space and improves the search efficiency compared to the NE method.

Given a trained CNN model and all edge devices with in total N PEs, our DSE methodology searches for Pareto CNN mappings to optimize the three objectives, mentioned in Section 3. In Figure 3, we present the general structure of our multi-stage hierarchical DSE methodology. On the left, the K stages in our DSE workflow are depicted, and on the right a zoomed-in view of each stage is provided with the two rectangular boxes showing the two hierarchical levels per stage. We accelerate our DSE process by splitting it into K different stages, where K is the ceiling value of $log_2(N)$. At each stage, we perform a two-level DSE. In both levels, the NSGA-II GA is deployed to evolve a population of

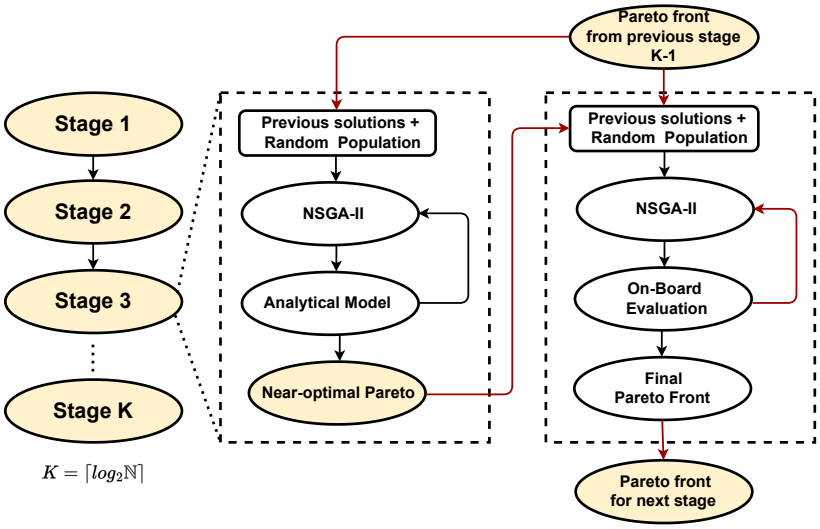

Fig. 3. The DSE Methodology workflow

CNN mappings over multiple generations to search for a Pareto front in terms of the targeted objectives. In the first DSE level, we use the analytical models, introduced in Section 3, inside the GA to approximate each objective function. In the second DSE level, we use real distributed CNN inference implementations generated by AutoDiCE (see Figure 1) for evaluation, thereby producing more accurate Pareto solutions as they are based on real (on-board) measurements.

At every DSE stage $k \in [1, \cdots, K-1]$, we search for optimal CNN mappings on 2^k target PEs. Figure 3 shows that to initialize the GA population at stage k, with $k > 1$, the Pareto optimal results found by the previous stage $k-1$ are used. By doing so, we can retain the information of Pareto CNN mappings in previous stages to improve the DSE convergence. Moreover, the second level DSE at each stage also uses the results from the first level of DSE to initialize its population. Finally, the output of the last DSE stage ($k = K$) provides the final Pareto-optimal solutions for N PEs.

5 Experimental Evaluation

In this section, we evaluate the search efficiency of our multi-stage hierarchical DSE methodology by conducting three DSE experiments and comparing the obtained experimental results in terms of the quality of the found solutions and how this quality changes over time during the DSE process (i.e., the search).

5.1 Experimental setup

In our three DSE experiments, we search for Pareto-optimal mappings of the popular ResNet-101 [8] CNN model onto a cluster of four edge devices. ResNet-101 has 344 layers with diverse types leading to an immense number of different CNN mappings, i.e., we have to perform the search in a vast design space. Therefore, ResNet-101 is a sufficiently representative model to apply our DSE methodology on and to demonstrate its merits. Our 4-device edge cluster consists of four NVIDIA Jetson Xavier NX development boards [13] that are connected via a Gigabit network switch. Each board has an embedded MPSoC featuring a 6-core CPU (NVIDIA Carmel ARMv8) and a GPU (Volta with 384 NVIDIA CUDA cores and 48 Tensor cores). Thus, we have 8 PEs in total in our edge cluster (4 boards with 1 CPU and 1 GPU per board). The On-Board Evaluation step in the second level of our DSE methodology (see Figure 3) measures and collects the CNN inference throughput, memory usage per device, and energy consumption per device over 20 CNN inference executions and represents them as average values over these 20 executions.

In the first DSE experiment, referred as 3s-2l-SPE, we utilize our multi-stage hierarchical DSE methodology as presented in Section 4 with 3 stages, 2 levels per stage, and the chromosome is encoded using our SPE method. In the second experiment, referred as 1s-non-SPE, we utilize a classical 1-stage, non-hierarchical DSE methodology based on the NSGA-II algorithm with our On-Board Evaluation as the fitness function and our SPE as the chromosome encoding method. In the third experiment, referred as 1s-non-NE, we utilize the same DSE methodology as in the second experiment but we replace SPE with the NE method mentioned in Section 4. In all experiments, every CNN layer can be mapped either onto a 6-core CPU or a GPU present in any of the aforementioned four board. The NSGA-II algorithm is executed with a population size of 100 individuals, a mutation probability of 0.2, and a crossover probability of 0.5. In each DSE experiment, we run the search for optimal mappings for 70 hours and compare the quality of solutions found within these 70 hours.

5.2 Experimental results

Figure 4 shows how the quality of the found mappings in terms of the three objectives, discussed in Section 3, improves during the search in the three DSE experiments. The results for each objective are plotted in a separate chart where the X-axis represents the search time in hours and the Y-axis represents the objective value in images per second (IPS) for the CNN inference throughput, in mega bytes (MB) for the maximum memory usage per edge device, and in joules per image (J/img) for the maximum energy consumption per edge device. Every point in a chart represents the best found mapping with respect to the objective at a given point in time.

The results in Figure 4 clearly indicate that the 1s-non-NE DSE gets easily stuck in dominance resistant solutions, which means that such DSE cannot find high-quality mappings even after hundreds of generations. In contrast, by

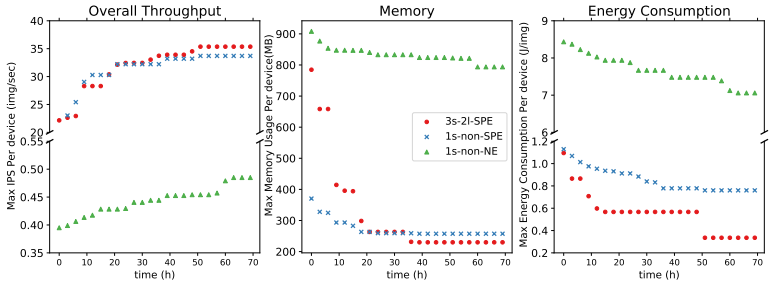

Fig. 4. Quality of found mappings during the three DSE experiments

replacing the common NE encoding method with our tailored SPE method, the search efficiency is significantly improved as shown in Figure 4 where the 1s-non-SPE DSE delivers high-quality mappings for the three objectives after 20 hours. This is because our SPE method ensures that only consecutive CNN layers will be mapped on a PE, thereby scaling down significantly the design space and allowing only exploration of mappings with reduced data communication among PEs. Such mappings are better than less restricted mappings allowed by the NE method.

Finally, comparing the 1s-non-SPE and 3s-2l-SPE results shown in Figure 4, we see that by introducing multiple stages and hierarchy in the DSE process, it is accelerate further in finding high-quality mappings. For example, after 40 hours of search time, our 3s-2l-SPE DSE delivers better mappings for the three objectives than the 1s-non-SPE DSE.

6 Conclusion

We have presented a novel multi-stage hierarchical DSE methodology for distributed CNN inference at the edge. To accelerate the DSE process and improve its efficiency, our DSE methodology combines analytical models with real on-board measurements to speedup the evaluations of individual design points and utilizes a tailored chromosome encoding method to effectively scale down the explored design space. The methodology has been experimentally evaluated by searching for optimal distributed mappings of the ResNet-101 CNN model onto an edge cluster of four NVIDIA Jetson Xavier boards. The experimental results show that our multi-stage hierarchical DSE methodology has significantly improved search efficiency in comparison to a classical one-stage, non-hierarchical DSE methodology which employs the commonly used, naive chromosome encoding method.

References

1. AutoDiCE: https://github.com/parrotsky/autodice
2. Bai, J., Lu, F., Zhang, K., et al.: Onnx: Open neural network exchange (2019), https://github.com/onnx/onnx
3. Deb, K., Pratap, A., Agarwal, S., Meyarivan, T.: A fast and elitist multiobjective genetic algorithm: Nsga-ii. IEEE Transactions on Evolutionary Computation **6**(2), 182–197 (2002). https://doi.org/10.1109/4235.996017
4. Deb, K.: Multi-Objective Evolutionary Algorithms, pp. 995–1015. Springer Berlin Heidelberg, Berlin, Heidelberg (2015)
5. Dillon, T., Wu, C., Chang, E.: Cloud computing: Issues and challenges. In: 24th IEEE International Conference on Advanced Information Networking and Applications. pp. 27–33 (2010)
6. Guo, Y.: A survey on methods and theories of quantized neural networks. arXiv preprint arXiv:1808.04752 (2018)
7. Hadidi, R., Cao, J., Ryoo, M.S., Kim, H.: Toward Collaborative Inferencing of Deep Neural Networks on Internet-of-Things Devices. IEEE Internet of Things Journal **7**(6), 4950–4960 (Jun 2020)
8. He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition. In: Conference on Computer Vision and Pattern Recognition. pp. 770–778 (2016)
9. Hou, X., Guan, Y., Han, T., Zhang, N.: Distredge: Speeding up convolutional neural network inference on distributed edge devices. ArXiv **abs/2202.01699** (2022)
10. Kang, et al.: Neurosurgeon: Collaborative intelligence between the cloud and mobile edge. ACM SIGARCH Computer Architecture News **45**(1), 615–629 (2017)
11. Loni, M., Sinaei, S., Zoljodi, A., Daneshtalab, M., Sjödin, M.: Deepmaker: A multi-objective optimization framework for deep neural networks in embedded systems. Microprocessors and Microsystems **73**, 102989 (2020)
12. Minakova, S., Sapra, D., Stefanov, T., Pimentel, A.D.: Scenario based run-time switching for adaptive cnn-based applications at the edge. ACM Transactions on Embedded Computing Systems (TECS) **21**(2), 1–33 (2022)
13. NVIDIA: Jetson Xavier NX (2020), https://developer.nvidia.com/embedded/jetson-xavier-nx
14. Pang, L.M., Ishibuchi, H., Shang, K.: Nsga-ii with simple modification works well on a wide variety of many-objective problems. IEEE Access **8** (2020)
15. Pimentel, A.: Exploring exploration: A tutorial introduction to embedded systems design space exploration. IEEE Design & Test **34**(1), 77–90 (2 2017)
16. Stahl, R., Zhao, Z., Mueller-Gritschneder, D., Gerstlauer, A., Schlichtmann, U.: Fully distributed deep learning inference on resource-constrained edge devices. In: International Conference on Embedded Computer Systems. pp. 77–90 (2019)
17. Teerapittayanon, S., McDanel, B., Kung, H.T.: Distributed deep neural networks over the cloud, the edge and end devices. In: 2017 IEEE 37th international conference on distributed computing systems (ICDCS). pp. 328–339. IEEE (2017)
18. Zeng, L., et al.: Coedge: Cooperative dnn inference with adaptive workload partitioning over heterogeneous edge devices. IEEE/ACM Transactions on Networking **29**(2), 595–608 (2020)
19. Zhao, Z., Barijough, K.M., Gerstlauer, A.: DeepThings: Distributed Adaptive Deep Learning Inference on Resource-Constrained IoT Edge Clusters. IEEE TCAD **37**(11), 2348–2359 (Nov 2018)
20. Zhou, L., et al.: Adaptive parallel execution of deep neural networks on heterogeneous edge devices. In: 4th ACM/IEEE Symposium on Edge Computing. p. 195–208 (2019)