IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received 3 December 2022, accepted 27 December 2022, date of publication 4 January 2023, date of current version 9 January 2023.

Digital Object Identifier 10.1109/ACCESS.2023.3234245

RESEARCH ARTICLE

PANCODE: Multilevel Partitioning of Neural Networks for Constrained Internet-of-Things Devices

FABÍOLA MARTINS CAMPOS DE OLIVEIRA [1], LUIZ FERNANDO BITTENCOURT [2], (Senior Member, IEEE), CARLOS ALBERTO KAMIENSKI [1], (Senior Member, IEEE), AND EDSON BORIN [2]

[1]Center of Mathematics, Computing and Cognition, Federal University of the ABC, Santo André, São Paulo 09210-580, Brazil
[2]Institute of Computing, University of Campinas, Campinas, São Paulo 13083-852, Brazil

Corresponding author: Fabíola Martins Campos de Oliveira (fabiola.oliveira@ufabc.edu.br)

This work was supported in part by CNPq under Grant 142235/2017-2, Grant 314645/2020-9, Grant 404087/2021-3, and Grant 159565/2022-7; in part by FAPESP under Grant 2013/08293-7 and Grant 2020/14771-2. This research is part of the INCT of the Future Internet for Smart Cities funded by CNPq under Grant 465446/2014-0, Coordenação de Aperfeiçoamento de Pessoal de Nível Superior—Brazil (CAPES)—Finance Code 001, and FAPESP under Grant 14/50937-1 and Grant 15/24485-9.

ABSTRACT The increasing number of Internet-of-Things (IoT) devices will generate unprecedented data in the upcoming years. Fog computing may prevent the saturation of the network infrastructure by processing data at the edge or within these devices. Consequently, the machine intelligence built almost exclusively on the cloud can be scattered to the edge devices. While deep learning techniques can adequately process IoT-massive data volumes, their high resource-demanding nature poses a trade-off for execution on resource-constrained devices. This paper proposes and evaluates the performance of the PArtitioning Networks for COnstrained DEvices (PANCODE), a novel algorithm that employs a multilevel approach to partition large convolutional neural networks for distributed execution on constrained IoT devices. Experimental results with the LeNet and AlexNet models show that our algorithm can produce partitionings that achieve up to 2173.53 times more inferences per second than the Best Fit algorithm and up to 1.37 times less communication than the second-best approach. We also show that the METIS state-of-the-art framework only produces invalid partitionings in more constrained setups. The results indicate that our algorithm achieves higher inference rates and low communication costs in convolutional neural networks distributed among constrained and exceptionally very constrained devices.

INDEX TERMS Convolutional neural networks, distributed systems, fog computing, graph partitioning algorithms, Internet of Things, performance evaluation.

I. INTRODUCTION

The expected significant increase in the number of devices connected to the Internet in the next few years will change how we execute applications [1], [2], [3]. The common practice is processing data generated by IoT devices in the cloud. However, the increasing number of devices may hinder this paradigm due to network saturation, increased delays, or both: application requirements may not be satisfied, and the overall quality of service may decay [4]. A possible solution is Fog Computing which processes data closer to the devices that produce them, i.e., routers, gateways, or on the devices themselves [5].

Some IoT device types produce multimedia data [6], [7], which can be adequately processed with deep learning techniques [8]. Convolutional Neural Networks (CNNs) are

The associate editor coordinating the review of this manuscript and approving it for publication was Cong Pu.

This work is licensed under a Creative Commons Attribution 4.0 License. For more information, see https://creativecommons.org/licenses/by/4.0/

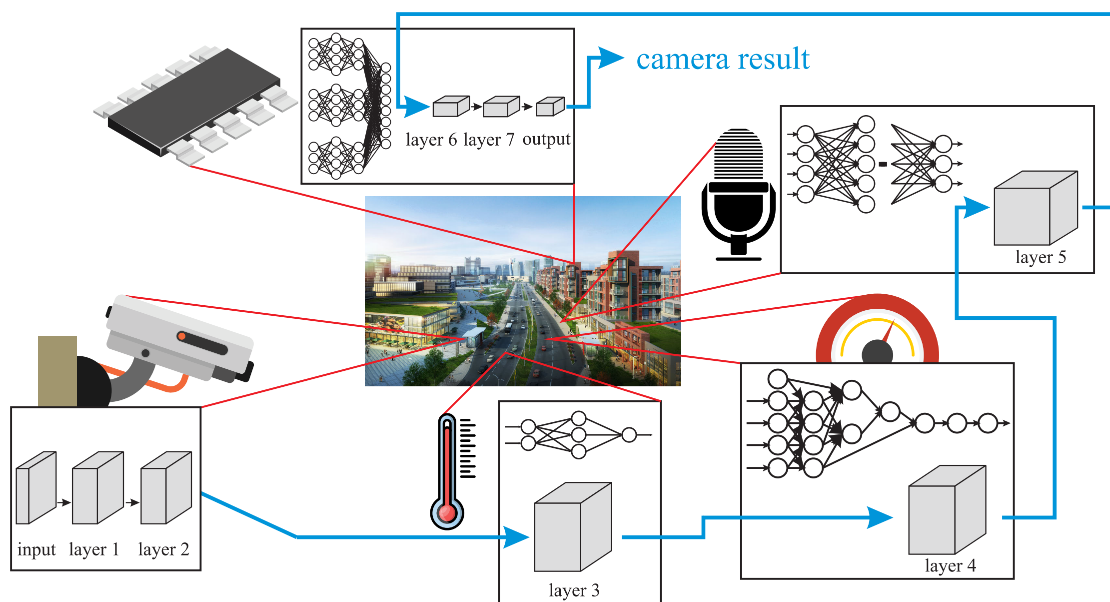

FIGURE 1. Example of distributing a DNN to execute its inference on multiple devices.

among the most popular deep learning techniques to process multimedia data due to their high-quality results, at the cost of increased demand for computational resources such as memory and CPU [9], [10], [11]. On the other hand, many IoT devices will be low cost and resource-constrained to achieve the expected billions of devices connected to the Internet [12].

There are two solutions to overcome the problem of executing CNNs on constrained IoT devices: reducing the CNN model [13] or partitioning the model for distributed execution on multiple devices [14]. The first solution might not be effective because the reduced CNN model may still require more resources than available in specific constrained devices. Therefore, in this paper, we partition the CNN model to fit the memory of constrained IoT devices for distributed processing. Fig. 1 illustrates the distribution of a DNN into multiple devices, where cameras in a smart city collect images to be processed and perform part of a DNN inference with their extra resources. After this step, the cameras send the data to nearby devices to distribute the DNN inference. These devices also perform their primary tasks, which may be the inference of other neural networks.

Existing frameworks that distribute a CNN on IoT devices limit the partitioning into layers [13], [15], [16]. However, this approach might lead to suboptimal results or memory-invalid partitionings, in which at least one partition needs more memory than the devices provide [14], [17]. General-purpose partitioning algorithms do not limit the partitioning into layers and can distribute the CNN execution automatically. However, we have also shown that they may produce suboptimal results and memory-invalid partitionings [14].

Recently, we proposed the Deep Neural Networks Partitioning for Constrained IoT Devices (DN^{2}PCIoT) algorithm for CNN partitioning into constrained IoT devices [14]. However, the effectiveness of the partitioning process comes with the price of a high execution time. To overcome the lower efficiency of DN^{2}PCIoT, this paper proposes the PANCODE algorithm to partition large general-purpose graphs faster, leveraging two DN^{2}PCIoT characteristics. DN^{2}PCIoT provides valid partitionings for very constrained setups and partitionings with larger inference rates or smaller amounts of communication than the state of the art. PANCODE primarily uses a multilevel approach, reducing the graph size gradually by grouping vertices, executing a partitioning algorithm in the smallest graph, before returning to the original graph gradually and refining the partitioning at each subgraph [18].

We partition the LeNet [19] and AlexNet [20] CNN models and compare PANCODE to the METIS state-of-the-art framework [18] and Best Fit [21]. Although these models are the first successful CNNs, they require fewer resources compared to more recent ones [22], making them more suitable for constrained devices. Our results show that PANCODE can improve up to 2173.53 times the inference rate or 1.37 times the communication messages. Also, METIS cannot provide memory-valid partitioning for very constrained devices. The source code of the algorithm implementation, the graphs for the CNNs, and the setups are available online [23].

The main contributions of our paper are:

- A novel multilevel algorithm to partition general-purpose dataflow graphs;
- A more flexible vertex grouping limitation, depending on the number of vertices in the dataflow graph and the number of devices in the setup;
- A more significant reduction in the number of edges and communication when grouping vertices;
- An always-valid partitioning for the smallest graph, which satisfies memory constraints since the algorithm beginning;

- A more flexible partitioning algorithm, which executes 1) one DN^{2}PCIoT epoch or 2) all DN^{2}PCIoT epochs, according to the number of subgraphs vertices and the number of devices; and
- A case study to validate PANCODE and experiment with two CNNs and several resource-constrained setups for inference rate maximization and communication minimization.

This paper is organized as follows: Section II presents the background in CNNs and their partitioning, while Section III discusses the related work. Section IV explains the PANCODE algorithm, while Section V details the CNN models, setups, and algorithms used in the experiments. Section VI discusses the experimental results, and Section VII points out future research directions. Finally, Section VIII summarizes our conclusions and future work.

II. BACKGROUND

This section comprises the background in CNNs, their representation and partitioning as a dataflow graph, and the problem definition.

A. CONVOLUTIONAL NEURAL NETWORKS

The idea behind CNNs comes from how the human brain processes vision [24], with some areas for recognizing the details of the scene captured by the eyes and others for recognizing the global scene. In CNNs, the convolution layers recognize the scene details, so that we can view every convolution layer neuron as the output of a brain neuron connected to only a tiny input region. Besides convolution layers, CNNs are also composed of pooling and fully connected layers. The pooling layers make the representations, *e.g.*, edges, shadows, and faces, invariant to translations and reduce their dimensionality. Finally, the fully connected layers recognize and classify the global scene. Essentially, CNNs learn increasingly complex representations at coarser resolutions along the layers.

We base the architecture of a CNN on specific characteristics of images, organizing the CNN layers in three dimensions: width, height, and depth. Fig. 2 shows an example of a CNN architecture with four layers. The first layer is the input, which usually has a depth of three for color images; the second layer is a convolution layer; the third layer represents a pooling layer; and the last one is a fully connected layer, corresponding to the CNN output. The output dimensions are 1 wide x 1 high, and the depth corresponds to the possible output classes. The first CNN layers are usually convolution and pooling. The former often requires more computation, especially the first layers, while the latter reduces the number of computations and parameters and, consequently, the memory for the subsequent layers. The fully connected layers are usually the layers that require more memory in a CNN.

Using a CNN is a two-phase process consisting of training and inference. The CNN generalizes input data during training tuning the parameters of the model. During the inference phase, the CNN predicts information on a new input based on the trained model and how well it can generalize data.

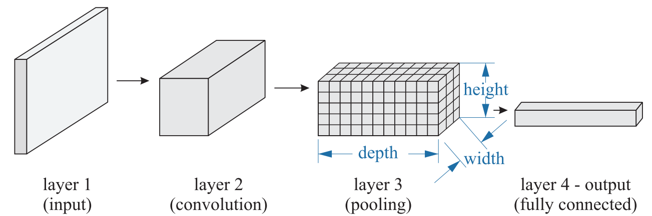

FIGURE 2. A CNN architecture with four layers: an input layer with a depth of three, a convolution layer, a pooling layer, and the fully connected output layer with a size of 1 wide x 1 high and the depth corresponding to the number of output classes (adapted from [24]).

B. PARTITIONING A CNN AS A DATAFLOW GRAPH

We can represent CNNs as directed acyclic dataflow graphs, which model the computation of a program through its data flow [25]. Vertices model the computations, and edges represent the data flow. Input edges represent data transfers to compute the corresponding vertex, whose results represent the output edges.

In our approach, a vertex may represent one or more CNN neurons and requires an amount of memory to store their results and parameters. The edges in the dataflow graph have an associated weight, which models the amount of data transferred to the vertices. Our dataflow graph representing a neural network shows the specification of the following per-layer data: the number of vertices in height, width, and depth, the layer type, and the amount of transferred data in bytes per inference required by each edge in each layer.

Fig. 3a shows a fully connected neural network represented as a dataflow graph, in which each vertex represents one neuron [14]. This network comprises three layers: an input layer with two vertices, a hidden layer with three vertices, and an output layer with one vertex. Each vertex of the input layer requires 4 bytes (B) to store the neuron input value, while each vertex of the hidden layer requires 12 B (4 B for the intermediate result and 8 B for the neuron parameters), and the vertex in the output layer requires 16 B (4 B for the final result and 12 B for the parameters). In this example, we do not use biases, inputs equal to 1 that improve model convergence. Each layer may have one bias unit.

Each vertex in the hidden layer performs 4 floating-point operations (FLOP) per inference, which correspond to multiplying the input values by the parameters, summing both values, and applying a function to this result. The vertex in the output layer performs 6 FLOP per inference, corresponding to three multiplications of parameters by input values, two sums, and the application of a function. Each edge transfers 4 B between each layer, corresponding to the intermediate values stored in each neuron.

Fig. 3b shows the same dataflow graph partitioned for distributed execution on two devices: device A (18 FLOP/second (FLOP/s) and 20 B of memory) and device

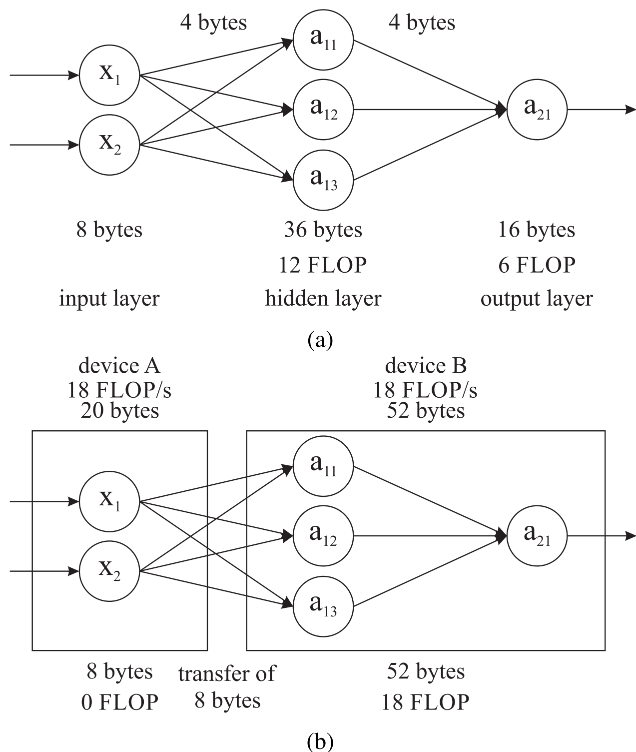

FIGURE 3. Examples of: (a) a fully connected neural network represented as a dataflow graph; and (b) how to partition it for execution on two devices [14].

B (18 FLOP/s and 52 B of memory). The link between them transfers 4 B/s, and the data transferred per inference is 8 B, which is the input data, although there are more than two edges in the dataflow graph. These additional edges are redundant, representing the transfer of the same data to the same partition, which often happens in partitioned CNNs.

When partitioning a CNN, we can optimize it for some objective function. If the objective function is the inference rate maximization, *i.e.*, the CNN throughput maximization, we calculate it as the minimum inference rate among all the devices and links, considering that they work in parallel. The inference rate of each device or link is, respectively, the computational or communication performance divided by the computational or communication load of each device or link. Thus, the CNN inference rate is:

$$\mathrm{iR} = \min\left\{\min\left[\left(\frac{\mathrm{compP}}{\mathrm{compL}}\right)_d\right], \min\left[\left(\frac{\mathrm{commP}}{\mathrm{commL}}\right)_{dq}\right]\right\}, \quad \forall\, d, q \in 1, \ldots, p \tag{1}$$

in which compP is the computational performance, compL is the computational load, commP is the communication performance, commL is the communication load, p is the number of devices, and dq is the link between devices d and q.

In the partitioning of Fig. 3b, device A does not perform any computation and, thus, does not limit the inference rate. Applying Equation 1, device A performs $18/0 = \infty$ inferences/s, device B performs $18/18 = 1$ inference/s, and the link between them supports $4/8 = 0.5$ inference/s. As the device or link with the lowest inferences per second limits the inference rate, the partitioning of Fig. 3b allows 0.5 inference/s. This partitioning is valid because each device fits the memory requirement of its respective partition, although it is possible to have an invalid partitioning when at least one device does not fit the memory requirement of its partition. In this paper, PANCODE attempts to find a valid partitioning that maximizes Equation 1.

A framework for task scheduling should synchronize data transfers among partitions to ensure correctness, thus guaranteeing input data completeness for all vertices. However, it may require extra time and reduce inference performance. Techniques such as retiming [26] can be applied to the partitionings to enforce synchronization. However, such a technique may increase the memory required to execute the CNN in a distributed form. If we maintain the memory provided by the devices, we may need to use more devices, and the amount of transferred data may increase. This increased data transfer may impact the inference rate if it becomes the execution bottleneck. Although this is an essential issue for deploying distributed CNNs on constrained IoT devices, here we compare our algorithm to state-of-the-art algorithms, which do not include the synchronization overhead.

C. PROBLEM DEFINITION

This section formally defines the partitioning problem as an objective-function optimization problem subject to constraints [14]. First, we define a function that returns 1 if an element e, which can be a neuron or a layer, is assigned to partition or device p and 0 otherwise:

$$\mathrm{partition}(p, e) = \begin{cases} 1, & \text{if } e \text{ is assigned to } p; \\ 0, & \text{otherwise.} \end{cases}$$

We define the partitioning problem as an optimization problem with an objective function subject to memory constraints:

$$\begin{aligned} &\text{optimize cost} \\ &\text{subject to } \sum_{n=1}^{\mathrm{N}} m_n \times \mathrm{partition}(p, n) \\ &\qquad + \sum_{l=1}^{\mathrm{L}} m_{\mathrm{sbp}_l} \times \mathrm{partition}(p, l) \le m_p, \\ &\qquad \forall\, p \in [1 \ldots P] \end{aligned}$$

in which cost is the objective function (detailed below), N represents the DNN number of neurons, m_n equals the memory required by neuron n, L is the number of layers of the DNN, m_{sbp_l} represents the memory required by the shared parameters and biases of layer l, m_p equals the memory that partition p can provide, and P is the number of partitions in the system. It is worth noting that partition(p, l) returns 1 if any neuron of layer l is assigned to partition or device p.

To minimize communication, we define a function that returns 1 if two elements are assigned to different partitions

or devices and 0 otherwise:

$$\text{diff}(e,h) = \begin{cases} 1, & \text{if } e \text{ and } h \text{ are assigned to different partitions or devices;} \\ 0, & \text{otherwise.} \end{cases}$$

Then, we define the communication cost, expressed in bytes per inference, as

$$\text{communicationCost} = \sum_{n=1}^{N} \sum_{z=1}^{\text{adj}(n)} \text{edgeWeight}_{nz} \times \text{diff}(n,z)$$

in which $\text{adj}(n)$ is the number of adjacent neurons of neuron n and edgeWeight_{nz} is the weight of the edge between neurons n and z, also expressed in bytes per inference.

To formally define the optimization problem, we rewrite the computational load of device d of Equation (1) as

$$\text{compL}_d = \sum_{n=1}^{N} \text{compL}_n \times \text{partition}(d,n)$$

and the communication load between devices d and q of Equation (1) as

$$\text{commL}_{dq} = \sum_{n=1}^{N} \sum_{z=1}^{\text{adj}(n)} \text{edgeWeight}_{nz} \times \text{diff}(n,z) \times \text{partition}(d,n) \times \text{partition}(q,z).$$

Here we model the CNN as a dataflow graph and partition it for distributed execution considering device memory and computational performance and the communication performance of the links between devices. We partition the CNN for inference execution to minimize the communication between devices or maximize the inference rate. As this problem is NP-complete [18], heuristic-based optimization is more appropriate to find partitionings with reduced communication or increased inference rate. PANCODE employs DN^2PCIoT [14], a heuristic-based algorithm that optimizes partitioning by swapping vertices between partitions or moving vertices among partitions.

III. RELATED WORK

This section discusses the related work in machine learning frameworks and partitioning algorithms and their relation to our proposal.

A. MACHINE LEARNING FRAMEWORKS

Some frameworks for executing neural networks distributed into several IoT devices have been proposed [13], [15], [27], [28], [29], [30]. Sze et al. [22] reviewed methods for efficiently executing DNNs, focusing on the inference phase, hardware platforms, and architecture for supporting DNNs. They discussed how to reduce the computational cost of DNNs by hardware design changes, algorithm changes, or both, providing metrics and design considerations when analyzing new DNN hardware and algorithmic designs. They also summarized the trade-offs between several hardware architectures and platforms.

FlexFlow [28] is a deep learning engine that automatically finds an optimized parallelization strategy using Sample-Operator-Attribute-Parameter (SOAP), which is a search space of parallelization strategies for DNNs that generalizes and overcomes previous approaches. FlexFlow comprises an incremental execution simulator that assesses different partitionings and a Markov Chain Monte Carlo search algorithm that explores the search space using the simulator information. The authors estimated the performance of some partitioning using a task graph that models both the DNN architecture and the cluster network topology. Nevertheless, FlexFlow does not consider memory constraints and optimizes only the execution time.

DeepThings [27] is a framework for inference distribution to constrained IoT devices that dynamically distributes and balances workload for the convolution layers, reduces execution latency and memory without accuracy loss, and increases inference rate. The partitioning occurs along the neural network data flow, repeating some computations in more than one device. The authors used few devices and a large amount of memory, which is unrealistic for very constrained devices. DeeperThings [13] is an evolution of this framework that fully distributes CNN inference, jointly optimizing memory, computation, and communication via Integer Linear Programming designs.

Kilcioglu et al. [30] proposed an algorithm to partition DNNs into constrained devices in the scenario of wireless Fog Computing aiming to reduce energy consumption. They partitioned the DNNs into layers or horizontally along the layers, optimizing communication and computation parameters and device workload. However, this algorithm replicates some computation at superior and inferior lines in the horizontal partitioning, which may impact the performance of constrained devices.

The Multi-fidelity DNNs [6] technique builds several neural networks with an increasing number of parameters. The size of each neural network is designed to match IoT devices with different computational resources and satisfy the heterogeneity in the IoT. However, there is some accuracy loss in every new neural network, which may not be acceptable under some conditions, such as fault detection in critical structures of Industry 4.0 [31].

DeepIoT [15] compresses CNNs, fully connected neural networks, and Recurrent Neural Networks by extracting redundant neurons. This compression can significantly reduce the DNN size, execution time, and energy consumption without loss of accuracy. However, this approach may not be sufficient because the DNN requirements may still be larger than the resources provided by a single constrained device.

The framework proposed by Li et al. [8] and the Heuristic Offloading Method (HOM) [29] offload some parts of the neural network code onto the cloud. While the framework of Li et al. [8] can offload layers, limiting the partitioning to this type, HOM [29] can only offload a complete inference task, not performing any partitioning.

TABLE 1. Main characteristics of machine learning approaches.

Approach	Prune neural network	Loss of accuracy	Offload to the cloud	Partitioning type	Automatic partitioning or pruning	Adequate account of shared parameters and biases
DeepIoT [15]	✓	×	×	-	✓	×
Li *et al.* [8]	×	×	✓	O	✓	×
DeepThings [27]	×	×	×	A (along the layers)	✓	×
Multi-fidelity DNNs [6]	✓	✓	×	-	×	×
FlexFlow [28]	×	×	×	SOAP	✓	×
HOM [29]	×	×	✓, and to the edge	-	✓	×
DeeperThings [13]	×	×	×	A	✓	×
Kilcioglu *et al.* [30]	×	×	×	OA	✓	×
AIDCC [32]	×	×	✓, and to the edge	-	-	×
iFaaSBus [33]	×	×	✓	-	-	×
DistrEdge [34]	×	×	to the edge	OA	-	×
PANCODE (proposed approach)	×	×	×	OAP	✓	✓

The Automatic and Intelligent Data Collector and Classifier (AIDCC) [32] is a framework that integrates IoT and deep learning, provides interpretability, offers transfer learning features, automatically collects data, visualizes extracted features, and detects diseases in pearl millet farmlands. AIDCC sends the collected data to a cloud server and IoT device. The authors developed the Custom-Net model that runs on the Cloud or IoT devices and predicts blast and rust diseases. They compared the Custom-Net results of transfer learning to the ones on state-of-the-art models such as Inception ResNet-V2 and VGG-19. Their model does not prune the neural network, does not present a loss of accuracy, and is comparable to state-of-the-art models. Custom-Net significantly reduces training time, aiding process automation and providing a low-cost framework. However, AIDCC does not partition the neural network and, thus, does not account for the memory required by the shared parameters and biases of CNNs.

iFaaSBus [33] is a lightweight framework that provides security and privacy to execute machine learning using IoT based on function as a service or serverless computing. iFaaSBus receives users' health data from IoT devices, sends the data to the cloud, and offloads the execution of machine learning models onto the cloud. The authors tested iFaaSBus to diagnose COVID-19 and the framework scalability, outperforming nonserverless computing. Additionally, they compared the accuracy, precision, recall, and F1-score of five machine learning models in COVID-19 diagnosis. As iFaaSBus does not prune the machine learning model, it does not present any loss in the machine learning metrics, including accuracy. Nevertheless, iFaaSBus does not respect memory constraints and, thus, does not account for the memory of shared parameters and biases when executing CNNs.

DistrEdge [34] is a method to distribute CNN inference execution to multiple IoT edge devices automatically. It considers device heterogeneity, network conditions, and specificities of CNNs in a deep reinforcement learning algorithm to partition the CNN into layers and the height of each layer and distribute the CNN inference execution. The authors tested offloading onto heterogeneous embedded Artificial Intelligence computing devices, such as NVIDIA Jetson, speeding up computations related to state-of-the-art methods, such as DeepThings and DeeperThings. DistrEdge does not prune the CNN; therefore, there is no loss of accuracy. However, as it does not consider memory constraints, DistrEdge does not account for the memory of shared parameters and biases in the CNN execution.

Table 1 summarizes the related work in machine learning approaches. We identified six main characteristics of these frameworks related to our algorithm. First, the approaches that prune the DNN may still require significant computational resources after pruning. This pruning may impact the DNN result accuracy, which is the second characteristic in Table 1. PANCODE does not prune the DNN; therefore, it does not impact the result accuracy. The third characteristic is execution offloading to the cloud. In our scenario, the devices may not always have an Internet connection. Thus, we need to execute the DNN only in the IoT devices. The partitioning type shows how the approach distributes the DNN if there is distributed execution, which is the fourth characteristic. It may be in the Operator (O), Attribute (A), Operator-Attribute (OA), Operator-Attribute-Parameter (OAP), and SOAP. PANCODE partitions DNNs using the OAP strategy, which allows partitionings into the layers, height, width, and depth of each layer, *i.e.*, the DNN neurons. In the fifth characteristic, the approaches may partition or compress the DNN automatically or manually, requiring the user to choose where to execute each DNN portion or how to reduce the DNN, respectively. PANCODE automatically partitions dataflow graphs, not demanding users with knowledge of computational systems and machine learning. Finally, the sixth characteristic is the adequate account of shared parameters and biases of CNNs, which may improve the partitioning result and enable partitioning into very constrained devices.

This characteristic enables the algorithms to calculate a precise amount of memory for each partition, thus allowing us to employ devices with a smaller amount of memory.

B. PARTITIONING ALGORITHMS

As the computation distribution may affect the inference performance and most IoT frameworks for neural networks constrain the partitioning, we can use general-purpose partitioning algorithms to achieve profitable partitionings. SCOTCH [35] is a framework that performs graph partitioning and static mapping based on the Dual Recursive Bipartitioning and the Fiduccia-Mattheyses algorithms, balancing the computational load and reducing communication. This framework does not handle memory constraints and, thus, may produce invalid partitionings. SCOTCH also does not factor redundant edges out of the cost computation.

Kernighan and Lin's algorithm (KL) [36] partitions graphs aiming to reduce communication and maintain partition balance by exchanging vertices between partitions. It can produce valid partitionings if the initial partitioning is valid. However, it cannot factor redundant edges out either.

METIS [18] is a framework that partitions large graphs and meshes and computes the orderings of sparse matrices. It uses the multilevel approach, which gradually groups the graph vertices to obtain smaller graphs, applies a partitioning algorithm to the smallest graph, and gradually returns to the original graph, refining the partitioning in each subgraph. METIS reduces communication and attempts to balance all the constraints, such as memory and computational load. However, METIS does not consider memory constraints nor factor redundant edges out of the cost computation.

Previously, we proposed Kernighan-and-Lin-based Partitioning (KLP) [17], an algorithm to partition CNNs into constrained IoT devices aiming for communication minimization and memory-valid partitionings. We showed that KLP provides partitionings with up to 4.5 times less communication than those provided by TensorFlow [37] and per-layer partitioning approaches. KLP factors redundant edges out of the cost computation and allows the movement of one vertex and the exchange of vertices between partitions. These characteristics are responsible for the improvement in communication.

We also recently proposed DN^{2}PCIoT [14], an algorithm that partitions CNNs into constrained IoT devices aiming for inference rate maximization or communication reduction. Besides the KLP characteristics, DN^{2}PCIoT accounts adequately for the memory required by the shared parameters and biases of CNNs and can start from partitionings obtained by other approaches. The former feature enables partitioning into very constrained devices. We showed that popular machine learning frameworks, such as TensorFlow, offer per-layer partitionings, which may lead to suboptimal or invalid results, which also happens in METIS.

Krylov and Friedman [38] proposed a methodology to balance bias current by automatically partitioning rapid single flux quantum (RSFQ) electronic circuits into blocks while minimizing the number of connections between blocks. They modified the Fiduccia-Mattheyses algorithm with the characteristics of RSFQ circuits and proposed a simulated-annealing partitioning, reducing the overall bias current. They used the quadratic placement algorithm, a geometric partitioning that performs a coarse placement, dividing the circuit into multiple blocks based on the coordinates of each vertex. However, the algorithm does not respect memory constraints nor factors redundant edges out of the cost computation.

Mt-KaHyPar [39] is a multilevel partitioning algorithm with a parallel implementation of all multilevel phases. The coarsening phase uses parallel community detection, the initial partitioning employs the recursive bipartitioning with work stealing, and the uncoarsening phase uses a scalable label propagation refinement. Mt-KaHyPar proposes the first fully-parallel direct k-way formulation of the Fiduccia-Matheyses algorithm as the partitioning algorithm in the initial partitioning and uncoarsening phases. Nevertheless, Mt-KaHyPar also does not consider memory constraints nor factors redundant edges out of the cost computation.

Blocking-Aware-Based Partitioning (BABP) [40] is an algorithm that partitions real-time dependent tasks into a homogeneous multi-core platform to reduce overall energy consumption and avoid deadline violations. BABP presents lower energy consumption and higher schedulability than popular bin-packing algorithms, such as Worst Fit Decreasing, Best Fit Decreasing, and Similarity-Based Partitioning. As a result, BABP leverages the available parallelism, assigning tasks to run in parallel on different cores and balancing the workload. Again, BABP does not produce partitions that respect memory constraints nor factors redundant edges out of the cost computation.

These general-purpose partitioning algorithms perform edge-cut partitionings, dividing the graph vertices into disjoint subsets. Another strategy is vertex-cut partitioning, which partitions the graph edges [41], [42]. However, it may replicate the vertices among partitions, which require more computation. Thus, considering the low computational performance of constrained IoT devices, this approach may not be adequate due to inference performance degradation.

JA-BE-JA-VC [41] is a vertex-cut partitioning algorithm that attempts to balance the partitioning while considering memory constraints. This approach needs vertex replicas, *i.e.*, computation replicas, and synchronization, which may involve more communication. For constrained IoT devices, these characteristics may decrease the inference rate of neural networks to a value lower than the application requirements. Additionally, JA-BE-JA-VC does not factor redundant edges out of the cost computation.

LeBeane et al. [42] modeled heterogeneity in the processing nodes of modern data centers to modify five online data ingress strategies. They aimed to optimize the execution time in heterogeneous data centers, improving the runtime of popular machine learning and data mining applications. They also considered partitioning algorithms that perform edge cuts and vertex cuts. When an application needs to have barriers for synchronization between different nodes, the authors

TABLE 2. Main characteristics of general-purpose partitioning approaches.

Approach	Memory constraints	Partition balance	Factor redundant edges out	Objective function
KL [36]	✓	Some unbalancing	×	Minimize communication
METIS [18]	×	Some unbalancing in the constraints	×	Minimize communication
SCOTCH [35]	×	Some unbalancing	×	Minimize communication
JA-BE-JA-VC [41]	×	✓	×	Balance partitions
LeBeane et al. [42]	×	Some unbalancing	×	Minimize execution time
KLP [17]	✓	×	✓	Minimize communication
DN^{2}PCIoT [14]	✓	×	✓	Maximize inference rate or minimize communication
Krylov and Friedman [38]	×	✓	×	Minimize communication and balance partitions
Mt-KaHyPar [39]	×	Some unbalancing	×	Minimize communication
BABP [40]	×	✓	×	Minimize energy consumption and avoid deadline violations
PANCODE (proposed approach)	✓	×	✓	Maximize inference rate or minimize communication

provide data proportionally to the computational performance of the nodes so that they have similar execution times when processing them. They also use proportions to define the amount of memory each partition requires, which does not impose a strict limit on the amount of memory of the partitions and may lead to invalid partitionings. Nevertheless, they do not factor redundant edges out of the cost computation.

Table 2 summarizes the main characteristics of the general-purpose partitioning approaches related to our algorithm. Memory treatment is vital in constrained IoT devices because they usually provide a small amount of memory. Thus, the algorithms need to consider memory constraints. PANCODE treats memory constraints, unlike most partitioning algorithms in the related work, which may lead to invalid partitionings. Partition balance attempts to equally partition the DNNs into similar sizes, usually resulting in good performance through load balancing. However, PANCODE allows unbalanced partitionings, widening the search space. KLP, DN^{2}PCIoT, and PANCODE are the only algorithms that can factor redundant edges out of the cost computation during the algorithm execution, which may reduce the final amount of communication by considering only the edges that represent the transfer of different data. The objective functions can be communication minimization, partition balance, execution time minimization, inference rate maximization, energy consumption minimization, or deadline violation avoidance. The inference rate maximization allows PANCODE to optimize throughput directly.

C. DISCUSSION

Our previous works [14], [17] investigated manual groupings of the LeNet neurons in each layer. These groupings allowed us to reduce the dataflow graph size and perform more experiments in a shorter time frame since the input was smaller than LeNet without any grouping. However, manually grouping vertices is time-consuming and prone to errors.

Here we propose a novel algorithm that uses the multilevel approach [18] to partition large neural networks directly into neurons, resulting in higher inference rates or less communication and maintaining the inference result related to a nonpartitioned execution. For convolution and pooling layers, we replicate the trained parameters when necessary, and for fully connected layers, we partition the parameter set together with the corresponding neurons. Our algorithm also provides memory-valid partitionings, which work for very constrained devices, and factors redundant edges out when grouping vertices, which can lead to partitionings with higher inference rates or less communication. Our setups provide more devices than the cited papers, and our algorithm does not increase the original number of computations.

IV. PANCODE

This section presents PANCODE, an algorithm that applies the multilevel approach to partition dataflow graphs into constrained IoT devices. We show an overview of PANCODE, our proposals for each phase of the multilevel approach, and the PANCODE algorithm.

Fig. 4 presents the three phases of the multilevel approach: coarsening, initial partitioning, and uncoarsening. The coarsening phase reduces the graph size by grouping the vertices into hypervertices according to the edges with the most significant values of communication. This process occurs gradually, generating subgraphs until only a few vertices remain. The initial partitioning phase partitions the coarsest graph produced in the previous step using, in our case, Best Fit as initial partitioning and our recently proposed DN^{2}PCIoT to improve it. Finally, the uncoarsening phase gradually ungroups the vertices, using DN^{2}PCIoT to refine the partitioning at each subgraph produced in the coarsening phase. Algorithm 1 depicts the PANCODE phases. We detail our contributions to each phase in the following subsections.

Algorithm 1 PANCODE Algorithm Overview

```
1:  function PANCODE(sourceG, nSubG)
2:      /* Coarsening phase */
3:      Build increasingly coarser nSubG graphs starting from the sourceG graph;
4:      /* Initial partitioning phase */
5:      Partition the coarsest subgraph produced in the previous step using the Best Fit algorithm;
6:      Improve the partitioning obtained in the previous step using DN2PCIoT;
7:      /* Uncoarsening phase */
8:      for each subgraph produced in the coarsening phase starting from the last but one coarsest graph do
9:          Map the partitioning obtained in the immediately coarser subgraph to the next and finer subgraph;
10:         Improve the partitioning obtained in the previous step using DN2PCIoT;
11:     end for
12:     return The best partitioning obtained by the algorithm;
13: end function
```

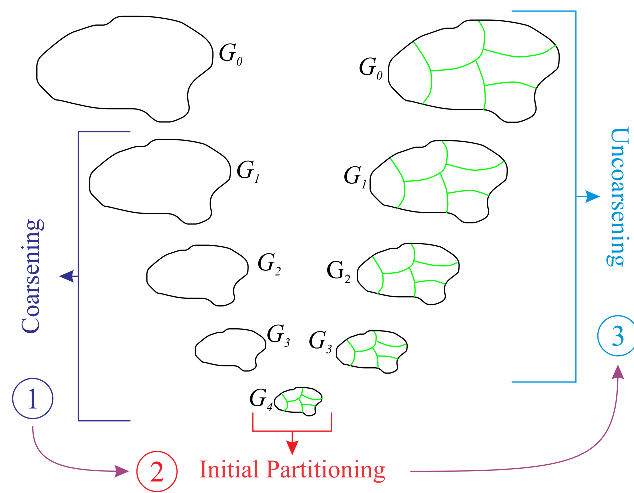

FIGURE 4. The three phases of the multilevel approach: coarsening, initial partitioning, and uncoarsening. G_0 is the source graph, G_1 to G_3 are increasingly coarser graphs, and G_4 is the coarsest graph. Adapted from the METIS manual [43].

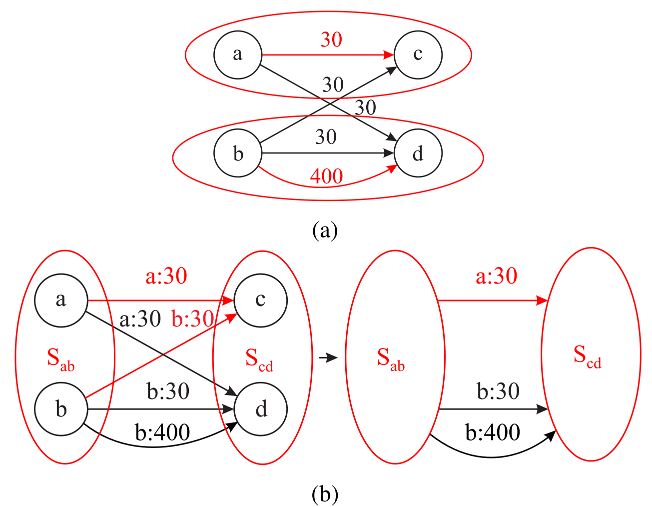

FIGURE 5. An example of (a) a heavy-edge matching and (b) a two-hop matching including our technique that factors redundant edges out of vertex grouping. We do not sum the edge weights to form only one edge between the grouped vertices S_{ab} and S_{cd}. Our proposal includes data about the source vertices so that when the algorithm groups vertices c and d, it removes the edges with repeated weights and sources. This process of factoring redundant edges out also happens in the heavy-edge matching.

A. COARSENING

In the coarsening phase, the multilevel approach gradually reduces the graph size, grouping vertices into hypervertices and creating subgraphs. We apply to PANCODE a modified METIS Heavy-Edge Matching (HEM) technique. For all ungrouped vertices, choose the edge with the largest weight and group the two vertices connected to it. Fig. 5a shows an example of HEM, which groups vertices a and c and vertices b and d according to the edge with the heaviest weight in each ungrouped vertex. We visit the vertices by their degree order so that every vertex can be grouped, like METIS. After HEM, if there are ungrouped vertices, we perform a two-hop matching in the graph to group two vertices if they were not grouped before in this subgraph and if they both have an edge that connects a vertex in common. Fig. 5b shows an example of a two-hop matching that groups vertices a and b and vertices c and d.

Unlike METIS, we build deterministic subgraphs, which require only one execution. Our subgraphs maintain some original edges to factor redundant edges out of the cost computation and vertex grouping. When METIS groups two vertices, if they have an edge to a vertex in common, METIS sums their weights and builds a subgraph with only one edge. On the other hand, we maintain these edges since they have different sources. The source of an edge is the vertex from which the edge leaves in a directed graph. If we group two vertices with edges with the same source and weight, we discard one edge because it is redundant. This process can significantly reduce the number of edges at a larger rate than METIS while factoring redundant edges out of the cost computation.

The two-hop matching example of Fig. 5b shows how we factor redundant edges out. Our algorithm groups vertices a and b into the grouped vertex S_{ab} and vertices c and d into the grouped vertex S_{cd}. We do not sum all the edge weights

between the grouped vertices S_{ab} and S_{cd} to form only one edge. Instead, our proposal includes data about the source vertices so that the algorithm removes edges with repeated weights and sources. We have two edges from the grouped vertex S_{ab} to the grouped vertex S_{cd} with the same weight (30) and same source (vertex a), representing the transfer of the same data. Thus, we use only one of these edges to form the next subgraph composed of the grouped vertices S_{ab} and S_{cd}. The exact process happens with the edge with weight 30 and source b, while the edge with weight 400 remains unchanged. This process of factoring redundant edges out also happens in HEM.

When we group vertices, we limit the vertex size so that the grouped vertices do not affect the balance among the vertices in the subgraphs, which may lead to memory-invalid partitionings, low inference rates, or high communication. METIS does not allow grouped vertices whose size is larger than a percentage of the sum of the size of all vertices. If there is only one constraint, METIS sets this percentage to 1.5%, and if there is more than one constraint, it is 7.5%. For CNNs, the sum of the size of all vertices is equivalent to the total memory required by the vertices, excluding the shared parameters and biases. We use this value to constrain the size of grouped vertices in the setups with 32 or more devices because they have devices with minimal memory. For the other setups, we limit the grouping by a percentage of the smallest amount of memory. This percentage depends on the number of source graph vertices and the number of devices. We adopt a percentage of 3.125% for graphs smaller than 700 vertices and setups between 4 and 11 devices and 25% in the other cases, based on several tests with different values for the LeNet and AlexNet models.

Next, METIS generates a fixed number of subgraphs, while PANCODE can receive the number of subgraphs as a parameter defined by the user for larger graphs. For smaller graphs, PANCODE also defines a fixed number of subgraphs. Changing the number of subgraphs influences the amount of time required by the algorithm and the partitioning result.

B. INITIAL PARTITIONING

In this phase, we partition the coarsest graph produced in the previous phase, applying an initial partitioning and a partitioning algorithm to refine it. METIS uses the graph-growing approach, which does not guarantee memory-valid partitionings. We use the Best Fit algorithm [21] to generate the initial partitioning and respect memory constraints. This algorithm assigns each vertex to the partition that fits it and contains the smallest amount of available memory after the vertex assignment. It assigns as many vertices as possible to the same partition, filling it before using another partition. Best Fit produces partitionings with less communication than the graph-growing approach for homogeneous setups.

METIS applies the Kernighan and Lin's algorithm based on the modification of Fiduccia and Mattheyses [44] to improve the partitioning produced by the graph-growing approach but stops if the Kernighan and Lin's algorithm produces the same result after 50 moves. PANCODE executes a modified version of DN^2PCIoT to improve the Best Fit partitioning. For each vertex v, search for the best vertex u that produces the most significant improvement in the cost function when swapping v for u. If the cost function remains unchanged after a defined number of vertices, stop the search and select the current best vertex u. This value, called swap stabilization, is an input parameter to PANCODE for larger graphs. Furthermore, for each vertex v, search for the best operation (swap or move). When searching for the best operation and best vertex or vertices that perform it, if the cost function produces the same result after another defined number of vertices, stop the search and choose the current best vertex v and its respective best operation. This value, called step stabilization, is an input parameter to PANCODE. With these modifications, DN^2PCIoT executes faster in PANCODE than the original version [14].

The approaches used by METIS, graph growing and Kernighan and Lin's algorithm, partition the source graph into all available devices. Our algorithm PANCODE applies Best Fit, which partitions the source graph into only the necessary devices according to their memory. Then, PANCODE executes DN^2PCIoT, which can discard more devices if the obtained partitioning results in a more significant inference rate or less communication. Thus, PANCODE maximizes the inference rate or minimizes communication and reduces the number of devices when possible.

C. UNCOARSENING

In the uncoarsening phase, the algorithm maps the partitioning obtained for the smallest subgraph to the subgraph of the next level, making its way back to the original graph. The algorithm refines the partitioning at each subgraph, which is possible because we have finer granularity in the larger subgraphs of higher levels. Thus, the algorithm can now assign vertices grouped before to different partitions.

Like METIS, we only consider vertices that present communication to vertices at a different partition during the refinements. METIS applies another version of Kernighan and Lin's algorithm in the uncoarsening phase. In this version, the algorithm executes only one epoch for each subgraph, so the partitioning algorithm executes faster. We perform a similar approach. However, to obtain a solution that executes faster but leverages DN^2PCIoT and based on validation tests, we execute all epochs if the number of devices is smaller than 12 or if the number of devices is smaller than 50 and the subgraph size is smaller than 700 vertices. This phase reuses the modifications of the initial partitioning, decreases the number of epochs, and only considers vertices that execute inter-partition communication. Thus, the algorithm executes faster.

D. PANCODE ALGORITHM

Algorithm 2 lists the pseudocode for PANCODE with the three phases explained in this section. First, we allocate a list of subgraphs and copy the original CNN graph into the

first position (Lines 2 and 3). Next, in the coarsening phase, a loop builds the coarser subgraphs, each one based on the previous subgraph (Lines 5–7). After that, in the initial partitioning phase, PANCODE applies Best Fit to the coarsest subgraph and saves the resultant partitioning to *bestP* (Line 9), which maps each graph vertex to one of the partitions. Then, PANCODE executes DN^{2}PCIoT to improve the Best Fit partitioning, saving the result to *bestP* (Line 10). The last phase is the uncoarsening, in which a loop runs through the other subgraphs until the original graph, executing DN^{2}PCIoT in each subgraph to improve the partitioning (Lines 12–14). Finally, PANCODE returns the best partitioning found (Line 15).

Algorithm 2 PANCODE Algorithm

```
1:  function PANCODE(sourceG, nSubG)
2:      subG[nSubG + 1];
3:      subG[0] ← sourceG;
4:      /* Coarsening phase */
5:      for n ← 1 to nSubG do
6:          subG[n] ← Coarsen(subG[n − 1]);
7:      end for
8:      /* Initial partitioning phase */
9:      bestP ← BestFit(subG[nSubG]);
10:     bestP ←DN²PCIoT(bestP, subG[nSubG]);
11:     /* Uncoarsening phase */
12:     for n ← nSubG − 1 to 0 do
13:         bestP ←DN²PCIoT(bestP, subG[n]);
14:     end for
15:     return bestP;
16: end function
```

V. METHODOLOGY

This section shows the CNN models, the setups for each model, the algorithms we execute to validate PANCODE, and how it compares to two literature algorithms.

A. CONVOLUTIONAL NEURAL NETWORK MODELS

We use two CNN models for the PANCODE validation: LeNet and AlexNet [19], [20]. While LeNet is a more lightweight model suitable for very constrained devices with 61 thousand parameters and 238 kB, AlexNet is a more robust model that requires more resources: 61 million parameters and 233 MB [45]. For both models, we group some neurons into one vertex to reduce the number of vertices in each dataflow graph, and we explain each grouping next.

For LeNet, we employ the versions LeNet 2:1 and LeNet 1:1 with different groupings [14], as shown in Fig. 6. The cubes represent the LeNet neurons, while the circles and ellipses represent the dataflow graph vertices. For LeNet 2:1 (Fig. 6a), we group the depth neurons in the same height and width into one vertex in each input, convolution, and pooling layer. We also group two neurons in width and two in height into one vertex, with each vertex in these layers containing four neurons of these dimensions plus the neurons in the respective depth. An exception is the last pooling layer, in which we group only the neurons in depth. In the fully connected layers, as the width and height have size one, we group every four neurons in the depth dimension into one vertex. The LeNet 2:1 model has 604 vertices, resulting in around four times fewer vertices than LeNet 1:1.

For LeNet 1:1 in Fig. 6b, we group only the depth neurons in the same height and width into one vertex in each input, convolution, and pooling layer. We model the fully connected layer neurons as one vertex each, resulting in a neural network with 2343 vertices. Table 3 shows the number of shared parameters and biases per layer and the memory and computation required by each vertex per layer, inference, and LeNet version. Here, the pooling layers present biases and trainable coefficients. Additionally, the fully connected layers require the most amount of memory, and the convolution layers require the most computation for both versions, including total and per-vertex amounts. In this table and hereafter, we represent the convolution layers by *C*, the pooling layers by *P*, and the fully connected layers by *FC*. While LeNet 2:1 groups more neurons, leading to faster partitioning since the graph size is smaller than LeNet 1:1, it constrains partitioning as it assigns the grouped neurons to the same partition. On the other hand, LeNet 1:1 allows more flexible partitionings as the algorithm assigns these neurons to different partitions. Partitioning two LeNet versions aim to verify if an expert-designed pregrouping leads to better results than a fully automatic grouping.

Fig. 7 presents the dataflow graph for AlexNet with 65,916 vertices, which is four layers deeper and has 28 times more vertices than LeNet 1:1. For each input, convolution, and pooling layer, we group the depth neurons in the same height and width into one vertex, as for LeNet. We model the fully connected layer neurons as one vertex each, similar to the LeNet 1:1 grouping. Table 4 shows the number of shared parameters and biases for each layer and the memory and computation required by each vertex in each layer per inference. Again, the fully connected layers require the most amount of memory, and the convolution layers require the most amount of computation, both total and per-vertex amounts.

B. DEVICE CHARACTERISTICS

LeNet uses five setups with four device models that gradually constrain memory, computation, and communication. They belong to the most constrained microcontroller big class of the Terminology for Constrained-Node Networks, between classes 0 and 4 (from 10 KiB to 1000 KiB of RAM) [50]. Table 5 shows the maximum number of devices allowed in each experiment, the device model name inspiring the experiments, the device memory, the device estimated computational performance, and the communication capacity between devices. The memory comes from the datasheets [46], [47], [48], [49] and we estimate the computational performance by their clock speed and the number of cores. Finally, communication employs a shared wireless medium with connections of up to 300 Mbits/s. We estimate the communication

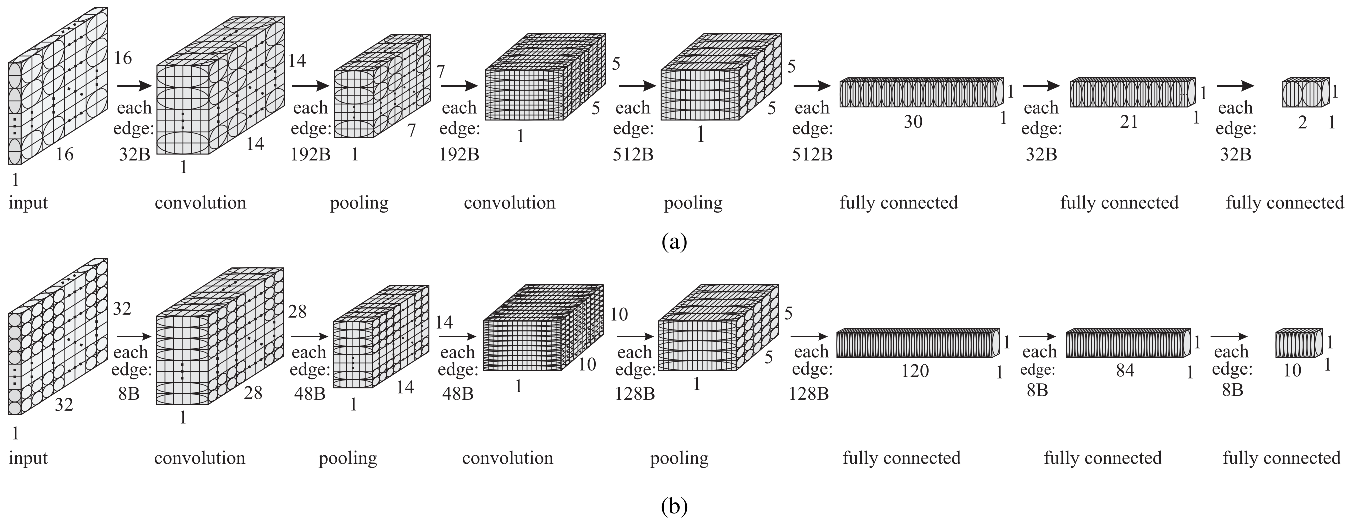

FIGURE 6. LeNet dataflow graph and vertex granularity used in our experiments. Each cube stands for a CNN neuron, while each circle is a vertex in the source dataflow graph. (a) LeNet 2:1: the LeNet model with 604 vertices, in which the width and height of each input, convolution, and pooling layer are divided by two, except for the last pooling layer, and the depth of the fully connected layers is divided by four. (b) LeNet 1:1: the LeNet model with 2343 vertices [14] (modified).

TABLE 3. Per-layer and per-vertex characteristics of each LeNet model used in this paper [14] (modified).

Characteristic	model	input	C1	P1	C2	P2	FC1	FC2	FC3
Memory of shared parameters and biases per layer (KiB)	both	0	1.22	0.09	11.8	0.25	0	0	0
Memory per vertex (B)	LeNet 1:1	8	48	48	48	48	3216	976	688
	LeNet 2:1	32	192	192	512	128	12864	3904	3440
Computation per vertex (FLOP) per inference	LeNet 1:1	0	306	36	765	96	51	240	168
	LeNet 2:1	0	1224	144	3060	96	204	960	840

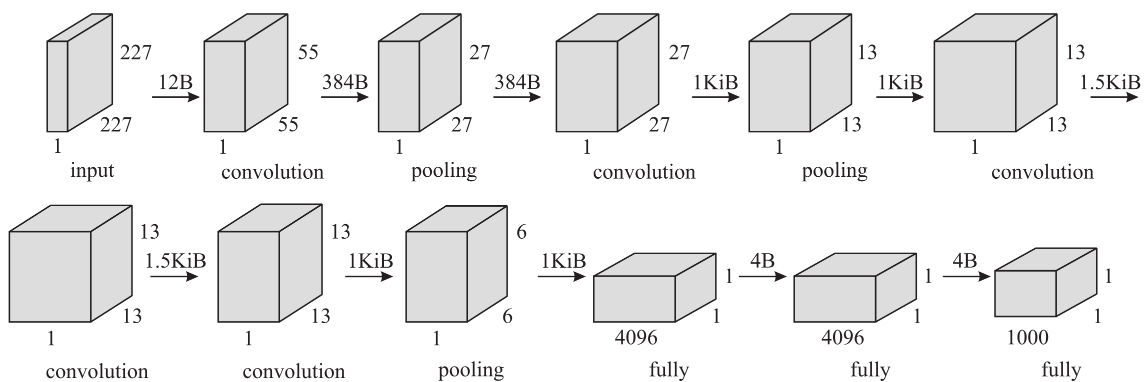

FIGURE 7. AlexNet dataflow graph.

performance depending on the maximum number of devices and assume that the communication links have a constant performance during the whole partitioning algorithm. Thus, the communication between devices for LeNet ranges from low (80 kbps) to reasonable (50 Mbps).

Table 6 shows nine setups for AlexNet with hypothetical devices belonging to classes 10 and 15 (from 4 MiB to 1024 MiB of RAM) of the least constrained general-purpose big class [50]. These setups offer less memory than standard embedded devices because, in many cases, these devices perform other tasks, and only a fraction of their memory is available for the CNN application. We gradually constrain the setups in memory and computational performance but use a high communication performance of 300 Mbits/s for all the setups. The communication links between devices present the same, constant performance

TABLE 4. Per-layer and per-vertex characteristics of the AlexNet model used in this paper.

Characteristic	input	C1	P2	C3	P4	C5	C6	C7	P8	FC9	FC10	FC11
Memory of shared parameters and biases per layer (MiB)	0	0.13	0	2.3	0	3.4	5.1	3.4	0	0	0	0
Memory per vertex (KiB)	0.01	0.38	0.38	1	1	1.5	1.5	1	1	36	16	16
Computation per vertex (kFLOP) per inference	0	69.9	0.86	5	2.3	5.1	7.7	7.7	2.3	18.4	8.2	8.2

TABLE 5. Setups for the LeNet experiments [14] (modified).

Number of devices	Device model	Device amount of RAM (KiB)	Device estimated computational performance (MFLOP/s)	Communication performance between devices (Mbit/s)
2	STM32F469xx [46]	388	180	50
4	Atmel SAM G55G [47]	176	120	25
11	STM32L433 [48]	64	80	2.7
56	STM32L151VB [49]	16	1.6	0.1
63	STM32L151VB [49]	16	1.6	0.08

TABLE 6. Setups for the AlexNet experiments.

Number of devices	Device amount of RAM (MiB)	Device estimated computational performance (GFLOP/s)	Communication performance between devices (Mbit/s)
2	183	312	300
4	76	156	300
8	46	78	300
16	31	39	300
32	21	20	300
40	8	10	300
47	7	8.75	300
54	6	7.5	300
63	5.5	6.25	300

for each experiment. AlexNet uses more powerful devices and communication, considering the future advancements in communication technology, such as 6G. Also, in the future, even constrained devices will provide more resources than current ones [50].

C. ALGORITHMS

We employ three algorithms for all the models, starting with Best Fit [21]. Also, we use the *gpmetis* from METIS [18], varying the number of partitions according to the setups, the number of partitionings to compute, the number of iterations for the refinement algorithms at each stage of the uncoarsening process, the maximum allowed load imbalance among the partitions, and the objective function of the algorithm. The latter can be edge-cut minimization or total communication volume minimization. Finally, we employ PANCODE, with different numbers of subgraphs, and choose the number that leads to the best cost after the Best Fit application in the initial partitioning phase. This best cost may be the highest inference rate when the algorithm objective is inference rate maximization or the smallest amount of communication when the algorithm objective is communication minimization. Then, we execute PANCODE entirely with the chosen number of subgraphs.

VI. EXPERIMENTAL RESULTS

This section discusses the results of the LeNet and AlexNet experiments for inference rate maximization and communication minimization and the coarsening phase results for AlexNet.

A. INFERENCE RATE MAXIMIZATION

The primary objective to partition a neural network when there is a data stream is usually the inference rate maximization, *i.e.*, throughput maximization. Fig. 8 shows the results in logarithmic scale for LeNet 2:1 and LeNet 1:1 when the objective function is inference rate maximization. We depict the cases in which METIS cannot produce valid partitionings as a red "x".

PANCODE achieves the highest results in 70% of the experiments, between 1.28 and 2.68 times higher than the second-best algorithm. Although the results of METIS are between 1.53 and 1.66 times higher than PANCODE in 30% of the experiments, it cannot produce valid partitionings for the most constrained setups. METIS does not limit the amount of memory required by each partition and does not account for the memory required by the shared parameters and biases of CNNs. On the other hand, PANCODE and Best Fit produce valid partitionings for all setups.

The Best Fit results for the 53- and 63-device setups are similar because they employ the same device model, maintaining the memory and computational performance. Also,

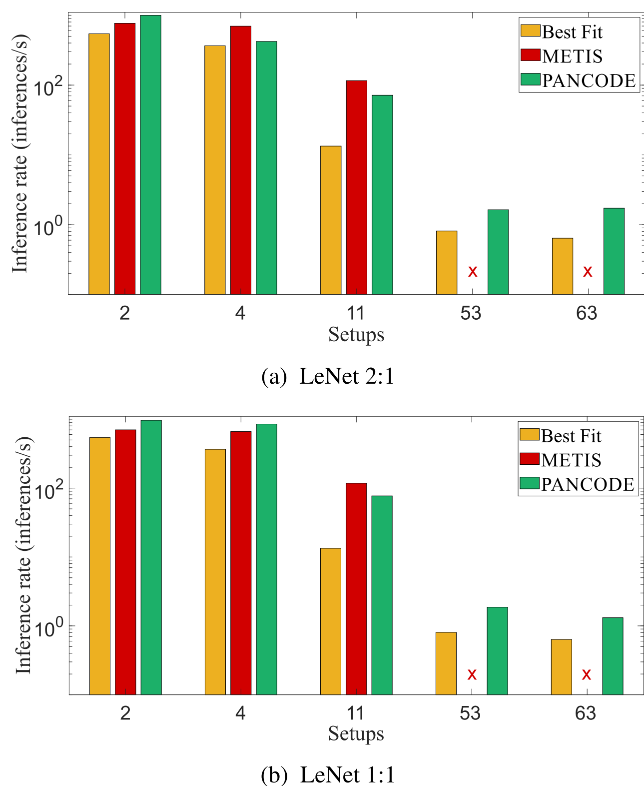

FIGURE 8. Inference rate maximization results for the LeNet models (higher is better).

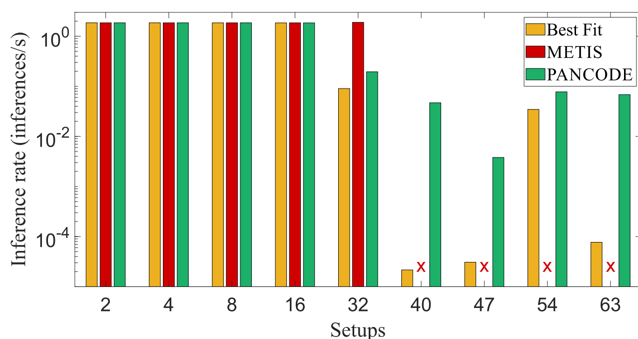

FIGURE 9. Inference rate maximization results for AlexNet (higher is better).

the 63-device setup has a reduced communication performance, which limits its performance since Best Fit fills one device before the next. PANCODE has the same influence since it uses Best Fit as initial partitioning in the coarsest graph.

For LeNet 2:1 and the 2-, 4-, and 11-device setups for LeNet 1:1, the inference rate of the initial partitioning in PANCODE is between 1.00 (equal) and 3.15 times smaller compared to Best Fit. Despite the initial partitioning value, PANCODE can lead to better results than Best Fit for all setups and achieve the best results in 75% of the experiments, suggesting that either the coarsening phase or the Best Fit initial partitioning limit its performance.

PANCODE achieves the highest results in 60% of the experiments with LeNet 1:1, between 1.08 and 2.02 times above LeNet 2:1. For the other 40%, the inference rate of LeNet 2:1 is between 1.03 and 1.31 times higher than LeNet 1:1. Therefore, the results suggest that we do not need expert-designed pregrouping such as LeNet 2:1 and can use the fully automatic grouping of PANCODE to achieve partitionings with high inference rates.

Fig. 9 shows the results in logarithmic scale for AlexNet when the objective function is inference rate maximization. Again, we normalize the inference rate, include the best inference rate achieved in each setup, and depict the METIS invalid partitionings as a red "x". Additionally, as LeNet 1:1 leads to higher inference rates in more setups than LeNet 2:1, we only use one AlexNet model, equivalent to LeNet 1:1.

PANCODE achieves higher or similar results in 88.9% of the experiments, between 1.00 and 2173.53 times higher than the second-best results, which come from Best Fit. For the least constrained setups with up to 16 devices, we expect the results to be similar because we use the same communication performance, and, as we double the number of devices in the experiments, we use half of the computational performance for each device, maintaining the total computational performance. However, for the setup with up to 32 devices, even maintaining the total computational performance, only METIS produces a result similar to the least constrained setups, while the result of Best Fit and PANCODE drops to 0.10 and 0.46 times the METIS result, respectively. For the most constrained setups with up to 40, 47, 54, and 63 devices, METIS again cannot produce any valid partitionings, unlike PANCODE and Best Fit. In this case, PANCODE achieves results between 2.24 and 2173.53 times higher than Best Fit.

B. COMMUNICATION MINIMIZATION

Even when there is a data stream, another objective to partition a dataflow graph is communication minimization so that the application does not overload the communication network. Fig. 10 shows the results in logarithmic scale for LeNet 2:1 and LeNet 1:1 when the objective function is communication minimization. We depict the cases in which METIS cannot produce valid partitionings as a red "x".

PANCODE achieves the lowest results in 90% of the experiments, between 1.02 and 1.37 times lower than the second-best algorithm. For LeNet 1:1 and the 4-device setup in Fig. 10b, Best Fit is 1.11 times lower than PANCODE. In this experiment, the initial partitioning communication produced by Best Fit in PANCODE is 1.26 times higher than the Best Fit communication, suggesting that either the coarsening phase or Best Fit limit PANCODE. For the most constrained setups, METIS cannot produce any valid partitionings again for the reasons described in the inference rate maximization results. Again, PANCODE and Best Fit produce valid partitionings for all setups. The Best Fit performance for the setups up to 53 and 63 devices is similar due to the similarity

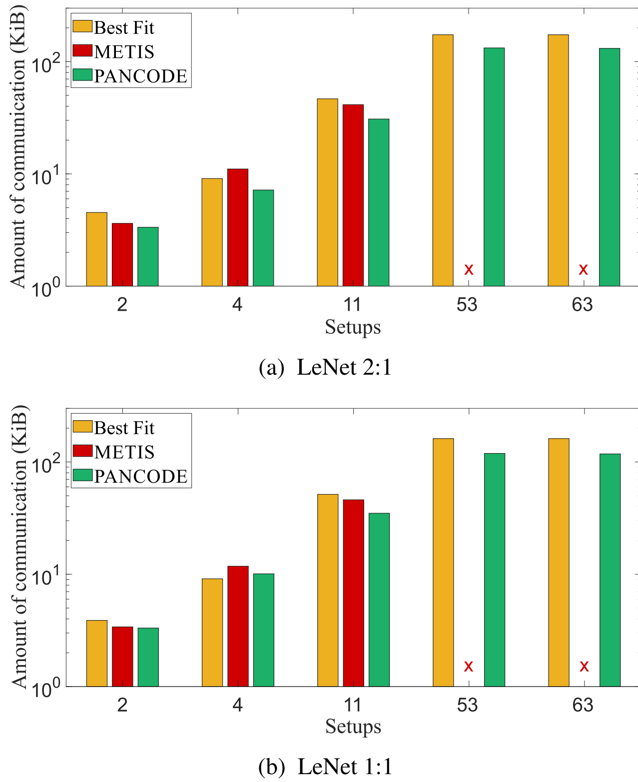

FIGURE 10. Results of communication minimization for the LeNet models (lower is better).

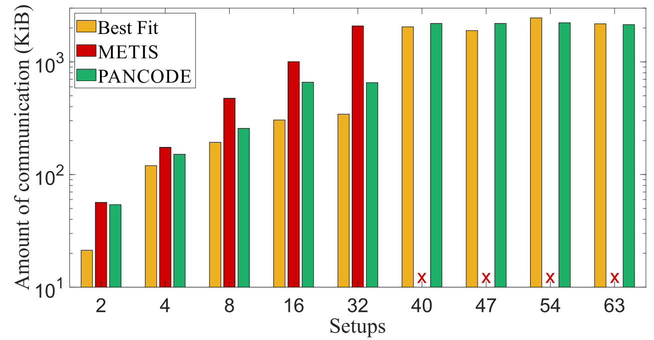

FIGURE 11. Results of communication minimization for AlexNet (lower is better).

between these setups and the Best Fit approach of filling one device before the next. The PANCODE results for the identical setups are also similar due to Best Fit as PANCODE initial partitioning.

PANCODE leads to the lowest result in 60% of the experiments with LeNet 1:1, between 1.01 and 1.11 times below LeNet 2:1. For the other 40%, LeNet 2:1 is between 1.13 and 1.40 times lower than LeNet 1:1. Although the LeNet 2:1 improvement is more significant than the LeNet 1:1 improvement, LeNet 1:1 achieves the lowest results in more setups than LeNet 2:1. Additionally, LeNet 2:1 is a more aggressive manual grouping of the LeNet model, time-consuming, and prone to errors. Therefore, the results for the proposed setups suggest that we can employ LeNet 1:1 for partitionings with low communication costs and limited impact on the results.

Fig. 11 shows the results for AlexNet when the objective function is communication minimization. Like the LeNet investigation, we also depict the METIS invalid partitionings as a red "x".

Unlike LeNet, Best Fit achieves the lowest results in 77.8% of the experiments, between 1.07 and 2.54 times below PANCODE, the second-best algorithm. Between 2 and 32 devices, METIS does not scale well as the difference between its results and the results of the other algorithms increase, reaching six times more communication activity than Best Fit in the setup with up to 32 devices. PANCODE is between 1.05 and 3.18 times lower than METIS for these setups, with the difference between them increasing as the number of devices in the setups increases, which shows that PANCODE scales better than METIS.

For the two most constrained setups, PANCODE achieves the lowest results, 1.02 or 1.10 times lower than Best Fit. Again, METIS cannot produce valid partitionings under the most constrained setups with up to 40 or more devices, unlike PANCODE and Best Fit. Although PANCODE only achieves the best results in some setups, it is better than Best Fit in half of the most constrained setups and 0.07 or 0.14 times higher in the other half. The results for the most constrained setups are similar due to the characteristics of the employed devices: the amount of memory per device drops 1.45 times between the setups up to 40 and 63 devices, and the computational performance drops 1.6 times. All the AlexNet communication minimization results also suggest that PANCODE is constrained by the coarsening phase or the use of Best Fit as initial partitioning, except the result for the most constrained setup. This suggestion is due to the communication of the initial partitioning produced by Best Fit in PANCODE in these setups, between 1.09 and 11.7 times higher than the communication of Best Fit.

C. COARSENING PHASE RESULTS

This section analyzes the coarsening phase of METIS and PANCODE due to the results in the previous subsections, which indicate this phase as one of the potential bottlenecks for PANCODE. Here the metrics are the number of subgraphs generated in the coarsening phase and the number of vertices and edges produced in the last subgraph, *i.e.*, the smallest, coarsest subgraph. Fig. 12 shows the number of subgraphs produced by METIS, PANCODE using the same number of subgraphs as METIS (called PANCODE-METIS), and PANCODE for inference rate maximization of AlexNet. While the results for METIS and PANCODE-METIS are the same, we define the number of subgraphs generated by PANCODE according to the subgraph that leads to the best initial cost (largest inference rate or lowest amount of communication), as explained in Subsection V-C. METIS creates seven to nine subgraphs, while PANCODE creates more subgraphs than METIS in 77.8% of the setups, between 6 and

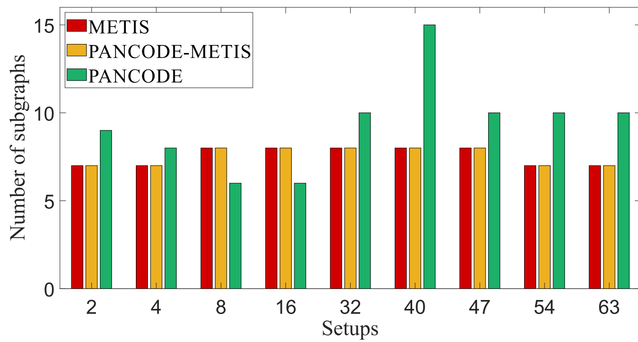

FIGURE 12. Number of subgraphs produced in the coarsening phase in METIS and PANCODE for AlexNet and inference rate maximization. PANCODE-METIS represents the results for PANCODE using the same number of subgraphs as METIS.

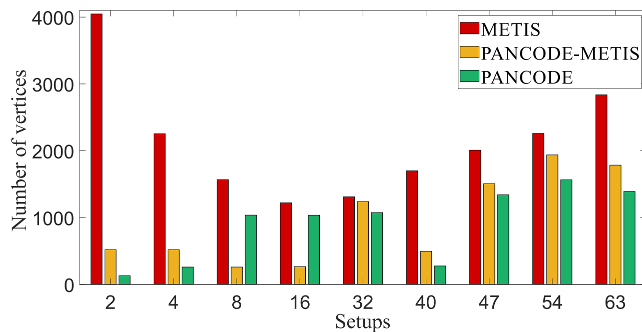

FIGURE 13. Number of vertices produced in the coarsening phase in METIS and PANCODE for AlexNet and inference rate maximization. PANCODE-METIS represents the results for PANCODE using the same number of subgraphs as METIS.

15 subgraphs. When PANCODE generates more subgraphs, there is a trend for the last subgraph to present fewer vertices and edges as METIS inspires the PANCODE approach to coarsen graphs.

Fig. 13 shows the number of vertices in the last subgraph produced by METIS, PANCODE-METIS, and PANCODE for inference rate maximization of AlexNet. PANCODE-METIS and PANCODE reduce the number of vertices related to METIS for all the setups, between 1.06 and 7.80 times and 1.18 and 31.13 times, respectively. PANCODE also reduces the number of vertices related to PANCODE-METIS, except in the 8- and 16-device setups, in which PANCODE generates fewer subgraphs than PANCODE-METIS and, thus, tends to show smaller reductions in the graph size.

Fig. 14 shows the number of edges in the last subgraph produced by METIS, PANCODE-METIS, and PANCODE for inference rate maximization of AlexNet. PANCODE-METIS reduces the number of edges between 1.44 and 12.34 times related to METIS in 55.5% of the setups, and PANCODE reduces it between 1.13 and 28.52 times related to METIS in 44.4% of the setups. In the setups in which PANCODE produces more edges than METIS, PANCODE creates fewer subgraphs than METIS only for the setup with up to 16 devices. In this setup, we expect PANCODE to

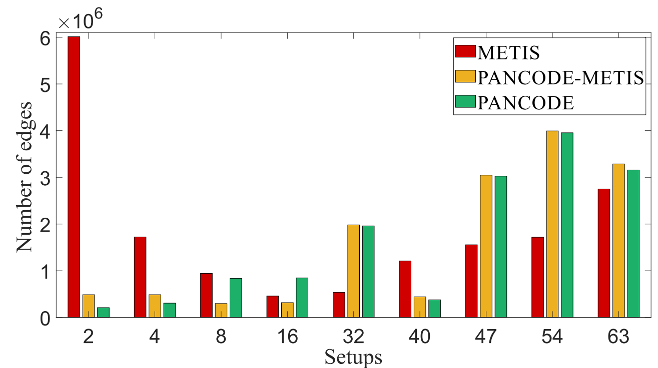

FIGURE 14. Number of edges produced in the coarsening phase in METIS and PANCODE for AlexNet and inference rate maximization. PANCODE-METIS represents the results for PANCODE using the same number of subgraphs as METIS.

have more edges in the last subgraph. The more significant number of edges in the last subgraph may explain the only poor result of PANCODE related to METIS in the setup with up to 32 devices in inference rate maximization in Subsection VI-A. We cannot compare the number of edges for the setups with up to 40 or more devices because METIS cannot produce valid partitionings for these setups. Additionally, in these setups, the large number of edges indicates that PANCODE cannot match many edges, find a significant number of redundant edges, or both.

For AlexNet with communication minimization, we have a similar pattern, with PANCODE producing up to 11 subgraphs and fewer subgraphs (six) than METIS (eight) only for the setup with up to four devices. PANCODE produces partitionings with less communication than METIS in this experiment for all setups. For LeNet, the same pattern appears: PANCODE generates more subgraphs for the setups with up to 11 or more devices and the same number or fewer subgraphs for the other setups. The number of vertices and edges in the last subgraphs produced by PANCODE is always smaller than the ones produced by METIS when using LeNet. These results show the effectiveness of the PANCODE coarsening phase in reducing the number of vertices and edges of the input graphs. Additionally, they indicate that we can improve the coarsening phase by reducing the number of edges in the generated subgraphs.

D. DISCUSSION

PANCODE generates partitionings with the highest inference rates for both LeNet and AlexNet in 78.9% of the experiments, up to 2173.53 times higher than the second-best algorithm (METIS or Best Fit). For smaller CNNs such as LeNet and the constrained devices of our setups, PANCODE provides a significant inference rate, which should meet the requirements of most applications, for instance, real-time surveillance. On the other hand, with AlexNet and constrained devices, PANCODE achieves inference rates that may not meet the requirements of some applications. In our scenarios, we envision a future in which the cities contain

billions of tiny, resource-constrained devices [12]. These devices perform simple tasks, and part of their resources can be used for other applications, for instance, CNN inference. Even though the most constrained setups achieve low inference rates, when we gather several setups with the same configuration and consider a large number of devices within a city, we may have appropriate resources to meet the applications inference rate requirement. Therefore, we maximize the inference rate by distributing its execution, and our results suggest that PANCODE is a proper algorithm to partition CNNs among constrained and exceptionally very constrained devices for inference rate maximization.

Communication minimization may not be an application requirement considering the same scenarios as inference rate maximization and data streams. However, an application is advantageous if it does not overload the communication network. Our results show that the partitionings generated with PANCODE achieve the lowest communication cost for small neural networks and larger CNNs with constrained devices. For larger CNNs and less constrained devices, PANCODE does not increase the communication cost substantially related to the best approach, which is Best Fit. These results indicate that PANCODE does not overload the communication network for exceptionally constrained setups. Thus, PANCODE is appropriate for partitioning small neural networks or distributing execution to exceptionally constrained setups in IoT and Fog Computing scenarios when the objective function is communication minimization.

When we partition LeNet for communication minimization, we have between 3.3 KiB and 132.6 KiB of total communication per inference among all the devices. For AlexNet, the total communication ranges from 54.1 KiB to 2.2 MiB per inference. For the real-time surveillance application, in which we need at least 24 frames per second, the data rate would be between 652.8 kbps and 1 Mbps for LeNet and 443.2 kbps and 18.5 Mbps for AlexNet. The LeNet setup devices comprise ARM processors, which support transfer rates up to 12 Mbit/s, and the AlexNet setups contain devices that support Wi-Fi 5 and 6. Thus, although the setups contain constrained devices, they can communicate at a sufficient data rate in our experiments.

We evaluate our previous manual grouping approach by comparing LeNet 2:1 to LeNet 1:1, a less aggressive grouping. We show that the automatic grouping performed by PANCODE leads to the highest inference rates or lowest amounts of communication for LeNet 1:1 in most setups. Thus, the LeNet results suggest that the PANCODE automatic grouping leads to more profitable results requiring less time and being error-free related to manual groupings.

Finally, we show data about the coarsening phase of the AlexNet inference rate maximization to analyze the advantages of our automatic grouping approach and point out how we can improve it. As explained in Subsection IV-A, PANCODE tends to create subgraphs with fewer edges than METIS due to its higher-degree matching and the factoring of redundant edges out of the cost computation. For most experiments, PANCODE can create smaller subgraphs than METIS. However, PANCODE cannot match many edges, find a significant number of redundant edges, or both for the setups with up to 32, 47, 54, and 63 devices. The last subgraph of PANCODE contains fewer vertices than the last subgraph created by METIS for all the experiments. Therefore, the automatic grouping of PANCODE effectively creates smaller subgraphs than METIS with less communication and a similar inference rate in the partitionings. To conclude our analysis, we can use PANCODE to partition dataflow graphs of different domains into any device, not only neural networks and constrained devices.

VII. FUTURE RESEARCH DIRECTIONS

We identify four main research lines to continue this study. First, we can perform more experiments to evaluate which phase of the multilevel approach should be improved (coarsening, initial partitioning, or uncoarsening). The coarsening phase tends to show the most significant impact on the partitioning result since coarsening too many vertices or vertices that present too many computations may constrain the possible configurations for partitions. In coarsening, we can adjust the number of subgraphs produced in this phase to avoid constraining the partitioning too much. The execution order of the matching techniques can be changed, performing two-hop matching before the heavy-edge matching. We can also propose other techniques to group vertices, for instance, based on constraining the vertex grouping according to the result of the initial partitioning applied to the source graph, i.e., performing an initial partitioning before coarsening.

We can use other algorithms to perform the initial partitioning, such as Worst Fit, First Fit, and Greedy [14]. In uncoarsening, we can change the conditions so that DN^{2}PCIoT executes more or all epochs at each subgraph refinement.

Besides the classical models tested in this paper, we can also perform experiments with different CNN models, such as specific models for constrained devices and recent models requiring more resources for execution. Although CNNs specially designed for constrained devices require fewer resources than regular CNNs, they still may require more resources than a very constrained device provides. Thus, partitioning CNNs designed for constrained devices is also necessary. Examples of specific models for constrained devices are SqueezeNet [51], MobileNet [52], and ShuffleNet [53]. More recent models comprise Channel Boosted CNN [54], [55], Convolution Block Attention Module [56], and Squeeze-and-Excitation Net [57].

We can test these CNNs with heterogeneous and more constrained setups, mixing devices from the most constrained microcontroller and least constrained general-purpose classes [50]. Finally, we can build a framework for task scheduling, fault tolerance – including rescheduling, online partitioning, and repartitioning –, and the inference aggregation of several setups. Thus, we can increase the application inference rate.

VIII. CONCLUSION

This paper proposes PANCODE, a multilevel algorithm that enhances our recently proposed DN^{2}PCIoT algorithm [14], improving its performance and enabling the partitioning of large graphs. We also propose modifications to DN^{2}PCIoT that runs within PANCODE.

We perform experiments using the LeNet and AlexNet CNN models for five and nine different setups, respectively, for inference rate maximization and communication minimization in PANCODE and compare them to the state-of-the-art METIS and the Best Fit algorithms. The results for inference rate maximization show that PANCODE leads to the highest inference rates for both neural networks in 78.9% of the experiments, up to 2173.53 times higher than the second-best algorithm. The results for communication minimization show that PANCODE achieves the lowest communication costs for small neural networks or very constrained setups and does not increase the communication cost substantially related to Best Fit for large neural networks. METIS cannot provide valid partitionings for both neural networks in the most constrained setups. Thus, our results suggest that PANCODE is an appropriate algorithm to partition CNNs among constrained and exceptionally very constrained devices.

For future work, we can perform experiments to evaluate which phase of the multilevel approach should be improved and start the improvement from it. We can also test different CNN models and heterogeneous and more constrained setups. Finally, we can build a framework for task scheduling, fault tolerance, and the inference aggregation of several setups to increase the application inference rate.

ACKNOWLEDGMENT

This work was supported in part by CNPq under Grant 142235/2017-2, Grant 314645/2020-9, Grant 404087/2021-3, and Grant 159565/2022-7; in part by FAPESP under Grant 2013/08293-7 and Grant 2020/14771-2. This research is part of the INCT of the Future Internet for Smart Cities funded by CNPq under Grant 465446/2014-0, Coordenação de Aperfeiçoamento de Pessoal de Nível Superior—Brazil (CAPES)—Finance Code 001, and FAPESP under Grant 14/50937-1 and Grant 15/24485-9.

REFERENCES

[1] Y. Mehmood, F. Ahmad, I. Yaqoob, A. Adnane, M. Imran, and S. Guizani, "Internet-of-Things-based smart cities: Recent advances and challenges," *IEEE Commun. Mag.*, vol. 55, no. 9, pp. 16–24, Sep. 2017, doi: 10.1109/MCOM.2017.1600514.

[2] M. Miraz, M. Ali, P. Excell, and R. Picking, "Internet of nano-things, things and everything: Future growth trends," *Future Internet*, vol. 10, no. 8, p. 68, Jul. 2018, doi: 10.3390/fi10080068.

[3] Cisco Systems. (Mar. 9, 2020). *Cisco Annual Internet Report (2018–2023)*. Cisco, San Jose, CA, USA. Accessed: Dec. 27, 2022. [Online]. Available: https://www.cisco.com/c/en/us/solutions/collateral/executive-perspectives/annual-internet-report/white-paper-c11-741490.html

[4] M. Merenda, C. Porcaro, and D. Iero, "Edge machine learning for AI-enabled IoT devices: A review," *Sensors*, vol. 20, no. 9, p. 2533, Apr. 2020, doi: 10.3390/s20092533.

[5] J. Lin, W. Yu, N. Zhang, X. Yang, H. Zhang, and W. Zhao, "A survey on Internet of Things: Architecture, enabling technologies, security and privacy, and applications," *IEEE Internet Things J.*, vol. 4, no. 5, pp. 1125–1142, Oct. 2017, doi: 10.1109/JIOT.2017.2683200.

[6] S. Leroux, S. Bohez, E. De Coninck, P. Van Molle, B. Vankeirsbilck, T. Verbelen, P. Simoens, and B. Dhoedt, "Multi-fidelity deep neural networks for adaptive inference in the Internet of Multimedia Things," *Future Gener. Comput. Syst.*, vol. 97, pp. 355–360, Aug. 2019, doi: 10.1016/j.future.2019.03.001.

[7] E. Baccour, N. Mhaisen, A. A. Abdellatif, A. Erbad, A. Mohamed, M. Hamdi, and M. Guizani, "Pervasive AI for IoT applications: A survey on resource-efficient distributed artificial intelligence," *IEEE Commun. Surveys Tuts.*, vol. 24, no. 4, pp. 2366–2418, Aug. 2022, doi: 10.1109/COMST.2022.3200740.

[8] H. Li, K. Ota, and M. Dong, "Learning IoT in edge: Deep learning for the Internet of Things with edge computing," *IEEE Netw.*, vol. 32, no. 1, pp. 96–101, Jan. 2018, doi: 10.1109/MNET.2018.1700202.

[9] P. Badjatiya, S. Gupta, M. Gupta, and V. Varma, "Deep learning for hate speech detection in tweets," in *Proc. 26th Int. Conf. World Wide Web Companion*, 2017, pp. 759–760, doi: 10.1145/3041021.3054223.

[10] L. Liu, H. Li, and M. Gruteser, "Edge assisted real-time object detection for mobile augmented reality," in *Proc. 25th Annu. Int. Conf. Mobile Comput. Netw.*, Aug. 2019, pp. 1–16, doi: 10.1145/3300061.3300116.

[11] N. Liu, Y. Xu, Y. Tian, H. Ma, and S. Wen, "Background classification method based on deep learning for intelligent automotive radar target detection," *Future Gener. Comput. Syst.*, vol. 94, pp. 524–535, May 2019, doi: 10.1016/j.future.2018.11.036.

[12] F. Pisani, F. M. C. de Oliveira, E. S. Gama, R. Immich, L. F. Bittencourt, and E. Borin, "Fog computing on constrained devices: Paving the way for the future IoT," in *Advances in Edge Computing: Massive Parallel Processing and Applications*, vol. 35, F. Xhafa and A. K. Sangaiah, Eds. Amsterdam, The Netherlands: IOS Press, 2020, pp. 22–60.

[13] 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, no. 4, pp. 600–624, Apr. 2021, doi: 10.1007/s10766-021-00712-3.

[14] F. M. C. D. Oliveira and E. Borin, "Partitioning convolutional neural networks to maximize the inference rate on constrained IoT devices," *Future Internet*, vol. 11, no. 10, p. 209, Sep. 2019, doi: 10.3390/fi11100209.

[15] S. Yao, Y. Zhao, A. Zhang, L. Su, and T. Abdelzaher, "DeepIoT: Compressing deep neural network structures for sensing systems with a compressor-critic framework," in *Proc. 15th ACM Conf. Embedded Netw. Sensor Syst.*, Nov. 2017, pp. 1–14, doi: 10.1145/3131672.3131675.

[16] J. Zhou, Y. Wang, K. Ota, and M. Dong, "AAIoT: Accelerating artificial intelligence in IoT systems," *IEEE Wireless Commun. Lett.*, vol. 8, no. 3, pp. 825–828, Jun. 2019, doi: 10.1109/LWC.2019.2894703.

[17] F. M. Campos de Oliveira and E. Borin, "Partitioning convolutional neural networks for inference on constrained Internet-of-Things devices," in *Proc. 30th Int. Symp. Comput. Archit. High Perform. Comput. (SBAC-PAD)*, Sep. 2018, pp. 266–273, doi: 10.1109/CAHPC.2018.8645927.

[18] G. Karypis and V. Kumar, "A fast and high quality multilevel scheme for partitioning irregular graphs," *SIAM J. Sci. Comput.*, vol. 20, no. 1, pp. 359–392, Aug. 1998, doi: 10.1137/S1064827595287997.

[19] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proc. IEEE*, vol. 86, no. 11, pp. 2278–2324, Nov. 1998, doi: 10.1109/5.726791.

[20] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet classification with deep convolutional neural networks," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 25, F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, Eds., 2012, pp. 1097–1105.

[21] E. G. Coffman, M. R. Garey, and D. S. Johnson, *Approximation Algorithms for Bin-Packing—An Updated Survey*. Vienna, Austria: Springer, 1984, pp. 49–106, doi: 10.1007/978-3-7091-4338-4_3.

[22] V. Sze, Y.-H. Chen, T.-J. Yang, and J. S. Emer, "Efficient processing of deep neural networks: A tutorial and survey," *Proc. IEEE*, vol. 105, no. 12, pp. 2295–2329, Dec. 2017, doi: 10.1109/JPROC.2017.2761740.

[23] F. M. C. de Oliveira. (2022). *MDN2*. GitHub. Accessed: Dec. 28, 2022. [Online]. Available: https://github.com/lmcad-unicamp/PANCODE

[24] Stanford University. (2018). *CS231n: Convolutional Neural Networks for Visual Recognition*. [Online]. Available: http://cs231n.github.io/convolutional-networks/

[25] M. Wolf, "Program design and analysis," in *Computers as Components*, 4th ed. Burlington, MA, USA: Morgan Kaufmann, 2017, ch. 5, pp. 221–319. [Online]. Available: http://www.sciencedirect.com/science/article/pii/B9780128053874000054

[26] C. E. Leiserson and J. B. Saxe, "Retiming synchronous circuitry," *Algorithmica*, vol. 6, nos. 1–6, pp. 5–35, Jun. 1991, doi: 10.1007/BF01759032.
[27] Z. Zhao, K. M. Barijough, and A. Gerstlauer, "DeepThings: Distributed adaptive deep learning inference on resource-constrained IoT edge clusters," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 37, no. 11, pp. 2348–2359, Nov. 2018, doi: 10.1109/TCAD.2018.2858384.
[28] Z. Jia, M. Zaharia, and A. Aiken, "Beyond data and model parallelism for deep neural networks," in *Proc. 2nd Conf. Syst. Mach. Learn. (SysML)*, 2019, pp. 1–13. [Online]. Available: https://mlsys.org/Conferences/2019/
[29] X. Xu, D. Li, Z. Dai, S. Li, and X. Chen, "A heuristic offloading method for deep learning edge services in 5G networks," *IEEE Access*, vol. 7, pp. 67734–67744, 2019, doi: 10.1109/ACCESS.2019.2918585.
[30] E. Kilcioglu, H. Mirghasemi, I. Stupia, and L. Vandendorpe, "An energy-efficient fine-grained deep neural network partitioning scheme for wireless collaborative fog computing," *IEEE Access*, vol. 9, pp. 79611–79627, 2021, doi: 10.1109/ACCESS.2021.3084689.
[31] A. Angelopoulos, E. T. Michailidis, N. Nomikos, P. Trakadas, A. Hatziefremidis, S. Voliotis, and T. Zahariadis, "Tackling faults in the industry 4.0 era—A survey of machine-learning solutions and key aspects," *Sensors*, vol. 20, no. 1, p. 109, Dec. 2019, doi: 10.3390/s20010109.
[32] N. Kundu, G. Rani, V. S. Dhaka, K. Gupta, S. C. Nayak, S. Verma, M. F. Ijaz, and M. Woźniak, "IoT and interpretable machine learning based framework for disease prediction in pearl millet," *Sensors*, vol. 21, no. 16, p. 5386, Aug. 2021, doi: 10.3390/s21165386.
[33] M. Golec, R. Ozturac, Z. Pooranian, S. S. Gill, and R. Buyya, "IFaaSBus: A security- and privacy-based lightweight framework for serverless computing using IoT and machine learning," *IEEE Trans. Ind. Informat.*, vol. 18, no. 5, pp. 3522–3529, May 2022, doi: 10.1109/TII.2021.3095466.
[34] X. Hou, Y. Guan, T. Han, and N. Zhang, "DistrEdge: Speeding up convolutional neural network inference on distributed edge devices," in *Proc. IEEE Int. Parallel Distrib. Process. Symp. (IPDPS)*, May 2022, pp. 1097–1107, doi: 10.1109/IPDPS53621.2022.00110.
[35] F. Pellegrini, "Distillating knowledge about SCOTCH," in *Proc. Dagstuhl Seminar-Combinat. Sci. Comput.*, U. Naumann, O. Schenk, H. D. Simon, and S. Toledo, Eds., 2009, pp. 1–12, doi: 10.4230/DagSemProc.09061.9.
[36] B. W. Kernighan and S. Lin, "An efficient heuristic procedure for partitioning graphs," *Bell Syst. Tech. J.*, vol. 49, no. 2, pp. 291–307, Feb. 1970, doi: 10.1002/j.1538-7305.1970.tb01770.x.
[37] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, and M. Devin, "TensorFlow: A system for large-scale machine learning," in *Proc. 12th USENIX Symp. Operating Syst. Design Implement.*, 2016, pp. 265–283. [Online]. Available: http://dl.acm.org/citation.cfm?id=3026877.3026899
[38] G. Krylov and E. G. Friedman, "Partitioning RSFQ circuits for current recycling," *IEEE Trans. Appl. Supercond.*, vol. 31, no. 5, pp. 1–6, Aug. 2021, doi: 10.1109/TASC.2021.3065287.
[39] L. Gottesbüren, T. Heuer, P. Sanders, and S. Schlag, "Scalable shared-memory hypergraph partitioning," in *Proc. Symp. Algorithm Eng. Exp. (ALENEX)*. Philadelphia, PA, USA: SIAM, 2021, ch. 2, pp. 16–30, doi: 10.1137/1.9781611976472.2.
[40] M. A. E. Sayed, E. S. M. Saad, R. F. Aly, and S. M. Habashy, "Energy-efficient task partitioning for real-time scheduling on multi-core platforms," *Computers*, vol. 10, no. 1, p. 10, Jan. 2021, doi: 10.3390/computers10010010.
[41] F. Rahimian, A. H. Payberah, S. Girdzijauskas, and S. Haridi, "Distributed vertex-cut partitioning," in *Distributed Applications and Interoperable Systems*, K. Magoutis and P. Pietzuch, Eds. Berlin, Germany: Springer, 2014, pp. 186–200.
[42] M. LeBeane, S. Song, R. Panda, J. H. Ryoo, and L. K. John, "Data partitioning strategies for graph workloads on heterogeneous clusters," in *Proc. Int. Conf. High Perform. Comput., Netw., Storage Anal.*, Nov. 2015, pp. 1–12, doi: 10.1145/2807591.2807632.
[43] G. Karypis. (2013). *METIS A Software Package for Partitioning Unstructured Graphs, Partitioning Meshes, and Computing Fill-Reducing Orderings of Sparse Matrices Version 5.1.0*. Accessed: Dec. 28, 2022. [Online]. Available: https://github.com/KarypisLab/METIS/tree/master/manual
[44] C. M. Fiduccia and R. M. Mattheyses, "A linear-time heuristic for improving network partitions," in *Proc. 19th Design Autom. Conf.*, 1982, pp. 175–181, doi: 10.1109/dac.1982.1585498.
[45] H. Abbas, I. Saha, Y. Shoukry, R. Ehlers, G. Fainekos, R. Gupta, R. Majumdar, and D. Ulus, "Special session: Embedded software for robotics: Challenges and future directions," in *Proc. Int. Conf. Embedded Softw. (EMSOFT)*, Sep. 2018, pp. 1–10, doi: 10.1109/EMSOFT.2018.8537236.
[46] STMicroelectronics. *STM32F469xx*. Datasheet. Accessed: Dec. 28, 2022. [Online]. Available: https://www.st.com/resource/en/datasheet/stm32f469ae.pdf
[47] Atmel. *SAM G55G/SAM G55J*. Datasheet. Accessed: Dec. 28, 2022. [Online]. Available: http://ww1.microchip.com/downloads/en/devicedoc/Atmel-11289-32-bit-Cortex-M4-Microcontroller-SAM-G55_Datasheet.pdf
[48] STMicroelectronics. *STM32L433xx*. Datasheet. Accessed: Dec. 27, 2022. [Online]. Available: https://www.st.com/resource/en/datasheet/stm32l433cc.pdf
[49] STMicroelectronics. *STM32L152x6/8/B Datasheet*. Accessed: Dec. 28, 2022. [Online]. Available: https://www.st.com/resource/en/datasheet/stm32l151vb.pdf
[50] C. Bormann, M. Ersue, A. Keränen, and C. Gomez. (Oct. 25, 2021). *Terminology for Constrained-Node Networks*. Internet Engineering Task Force. Accessed: Dec. 28, 2022. [Online]. Available: https://datatracker.ietf.org/doc/html/draft-bormann-lwig-7228bis-07
[51] F. N. Iandola, S. Han, M. W. Moskewicz, K. Ashraf, W. J. Dally, and K. Keutzer, "SqueezeNet: AlexNet-level accuracy with 50 x fewer parameters and less than 0.5 MB model size," 2016, *arXiv:1602.07360*.
[52] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, "MobileNets: Efficient convolutional neural networks for mobile vision applications," 2017, *arXiv:1704.04861*.
[53] X. Zhang, X. Zhou, M. Lin, and J. Sun, "ShuffleNet: An extremely efficient convolutional neural network for mobile devices," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, Jun. 2018, pp. 6848–6856, doi: 10.1109/CVPR.2018.00716.
[54] A. Khan, A. Sohail, and A. Ali, "A new channel boosted convolutional neural network using transfer learning," 2018, *arXiv:1804.08528*.
[55] A. Aziz, A. Sohail, L. Fahad, M. Burhan, N. Wahab, and A. Khan, "Channel boosted convolutional neural network for classification of mitotic nuclei using histopathological images," in *Proc. 17th Int. Bhurban Conf. Appl. Sci. Technol. (IBCAST)*, Jan. 2020, pp. 277–284, doi: 10.1109/IBCAST47879.2020.9044583.
[56] S. Woo, J. Park, J.-Y. Lee, and I. S. Kweon, "CBAM: Convolutional block attention module," in *Computer Vision—ECCV 2018*, V. Ferrari, M. Hebert, C. Sminchisescu, and Y. Weiss, Eds. Cham, Switzerland: Springer, 2018, pp. 3–19.
[57] Y. Hu, G. Wen, M. Luo, D. Dai, J. Ma, and Z. Yu, "Competitive inner-imaging squeeze and excitation for residual network," 2018, *arXiv:1807.08920*.

FABÍOLA MARTINS CAMPOS DE OLIVEIRA received the B.Sc. degree in control and automation engineering, the M.Sc. degree in mechanical engineering, and the Ph.D. degree in computer science from the University of Campinas, Campinas, Brazil, in 2012, 2015, and 2020, respectively. Currently, she is a Postdoctoral Researcher at the Federal University of ABC, Santo André, Brazil. Her research interests include federated learning, intelligent transportation systems, graph partitioning, distributed deep learning, fog computing, the Internet of Things, distributed computing, parallel computing, and high-performance computing.

LUIZ FERNANDO BITTENCOURT (Senior Member, IEEE) received the bachelor's degree in computer science from the Federal University of Parana, Brazil, and the master's and Ph.D. degrees from the University of Campinas (UNICAMP), Brazil. He is currently an Associate Professor at UNICAMP. He has been a Visiting Researcher with The University of Manchester, U.K., Cardiff University, U.K., and Rutgers University, USA. His research interests include the areas of resource management and scheduling in cloud, edge, and fog computing. He was awarded with the IEEE Communications Society Latin America Young Professional Award, in 2013. He was the TPC Co-Chair for IEEE/ACM UCC2018 and the Track Chair for CloudCom (2018–2019) and FiCloud, in 2017, 2018, and 2019. He served in several technical program committees. He also serves as an Associate Editor for the *IEEE Cloud Computing* magazine, the *Computers and Electrical Engineering* journal, the IEEE Internet of Things Journal, the *Journal of Network and Systems Management*, and the IEEE Networking Letters.

CARLOS ALBERTO KAMIENSKI (Senior Member, IEEE) received the B.S. degree in computer science from the Federal University of Santa Catarina, Florianópolis, Brazil, in 1989, the M.S. degree from the State University of Campinas, Campinas, Brazil, in 1994, and the Ph.D. degree in computer science from the Federal University of Pernambuco, Recife, Brazil, in 2003. He is currently a Full Professor in computer science at the Federal University of ABC (UFABC), Santo André, São Paulo, Brazil. His current research interests include the Internet of Things, smart agriculture, smart cities, network softwarization, and online social networks.

EDSON BORIN is currently an Associate Professor at the Institute of Computing, University of Campinas (Unicamp). Before joining Unicamp, in 2010, he was a Research Scientist at the Intel Laboratories, CA, USA, where he investigated and developed dynamic compilation techniques to enhance state-of-the-art HW/SW co-designed microprocessors, including automatic binary parallelization, dynamic binary translation, optimization techniques, and hardware support to accelerate single-threaded applications. He also applies his knowledge on modern computer architecture and compilers to investigate techniques to optimize existing scientific and engineering computing applications.

• • •