DONNA: Distributed Optimized Neural Network Allocation on CIM-Based Heterogeneous Accelerators

Mojtaba F. AlShams[1], Kamilya S. Smagulova[1], Suhaib A. Fahmy[1], Mohammed E. Fouda[2,†], and Ahmed M. Eltawil[1]
[1]CEMSE Division, King Abdullah University of Science and Technology, Thuwal 23955, Saudi Arabia
[2]Rain Neuromorphics, Inc., San Francisco, CA, 94110, USA
[†]Email: foudam@uci.edu

***Abstract*—The continued development of neural network architectures continues to drive demand for computing power. While data center scaling continues, inference away from the cloud will increasingly rely on distributed inference on multiple devices. Most prior efforts have focused on optimizing single-device inference or partitioning models to enhance inference throughput. Meanwhile, energy consumption continues to grow in importance as a factor of consideration. This work proposes a framework that searches for optimal model splits and distributes the partitions across the combination of devices taking into account throughput and energy. Participating devices are strategically grouped into homogeneous and heterogeneous clusters consisting of general-purpose CPU and GPU architectures, as well as emerging Compute-In-Memory (CIM) accelerators. The framework simultaneously optimizes inference throughput and energy consumption. It is able to demonstrate up to 4× speedup with approximately 4× per-device energy reduction in a heterogeneous setup compared to single GPU inference. The algorithm also finds a smooth Pareto-like curve in the energy-throughput space for CIM devices.**

***Index Terms*—Distributed Inference, Model splitting, Heterogeneous Hardware, Compute-in-memory, Heterogeneous Devices, GPU, CPU, CIM, ReRAM.**

I. Introduction

Deep Neural Network (DNN) models are known for their ability to solve complex problems with high accuracy. Their popularity led to the development of architectures of different variety and complexity, albeit at the cost of larger storage capacities and computational power. Graphics Processing Unit (GPU) with parallel processing took over Central Processing Unit (CPU) as a technology of choice due to its ability to accelerate operations such as Matrix-Vector Multiplication (MVM) which form the core of DNN. Nevertheless, general-purpose machines that are based on Von-Neumann architecture require constant data transfer between processing and memory, causing a "memory wall" problem and limiting the speed of data processing. For example, the state-of-the-art ViT-22B [1] and GPT-3 [2] models with so-called "attention" blocks have 22 and 175 billion trainable parameters, respectively. The NVIDIA H100 Tensor Core GPU [3], which is among the most powerful currently available GPUs, can not fit huge models such as GPT-3 for single device inference. It is also expected that emerging models will continue to grow in size and hardware accelerators will continue to suffer from limitations, as discussed above [4].

From a compute perspective, a promising approach to accelerate modern DNN models is to utilize novel hardware architectures such as emerging Compute-In-Memory (CIM) devices. Unlike traditional computing platforms, CIM does not mitigate the memory wall problem but solves it by combining storage and computation units. The memory elements used form a mesh of crossbar arrays that perform the MVMs using Kirchhoff's and Ohm's laws [5] [6]. Devices are designed to perform fast and energy-efficient operations, where ReRAM memory technology is one of the promising technologies to be used as a basic cell of CIM crossbar array. The design and performance of ReRAM-based accelerators such as ISAAC, PUMA, PipeLayer and others are reviewed in [7].

From an efficient inference performance perspective, distributing DNN across multiple devices is one of the ways to solve situations where the available resources cannot meet the targeted latency performance or when the model is too large to fit in a single device. Multiple works contributed to the solving and optimizing of distributed inference setups, mainly focusing on at-the-edge scenarios. One of them is DistrEdge which uses model parallelism to increase inference throughput [8]. The authors used greedy search and reinforcement learning to find the best layer-volumes and their split-parts. Their work included a variety of heterogeneous edge devices and different communication bandwidths (BW). Another work, PipeEdge algorithm [9], searches for best layer-splits of the model and tries to minimize the slowest stage in the inference pipeline. It aims to maximize the inference throughput while considering the edge device's memory constraints. Authors of Neurosurgeon [10] designed a framework that optimizes either the inference throughput or the energy consumption of a mobile-cloud system. However, It does not consider simultaneous optimization nor distributing on multiple devices as in DONNA. Neurosurgeon is fed with the model's layers characteristics and uses pre-generated prediction models to estimate the latency and energy costs of each possible partitioning.

This work proposes DONNA framework to find the optimal DNN model splits and optimal allocation to the available devices. It not only considers inference throughput optimization, as in most of prior work, but also considers energy optimization which is becoming an important factor in modern deployments. DONNA optimizes inference throughput and/or

energy consumption. The main contributions of the paper are: (1) The framework is designed to support heterogeneous nodes spanning edge (weak devices) or the cloud (strong devices), on homogeneous or heterogeneous devices' clusters; (2) Unlike previous works, this work can optimize throughput and/or energy by controlling a tuneable parameter α; (3) To the best of our knowledge, this is the first work which studies the effect of including CIM architectures in a distributed inference setup.

The rest of the paper is organized as follows: Section II presents DONNA, distributed optimal neural network allocation, how it operates, explains the search algorithm, and shows how it is different from [9]. Section III explains the experimental setup and details on how the simulation framework was constructed. Section IV discusses obtained results. Finally, Section V summarizes the most important points of this paper and provides conclusions and future work directions.

II. DONNA Framework

A. An Overview

DONNA exploits pipeline parallelism for distributed inference and splits model $\mathbb{M}$ with L layers across a homogeneous or heterogeneous set of devices comprised of CPU, GPU, and/or novel CIM platforms. Pipelining avoids collective synchronization and communication and improves the effectiveness of distributed systems. Our system aims to find the optimal model split to minimize the slowest pipeline stage. Furthermore, the proposed framework tries to minimize total energy consumption and to find a trade-off between the slowest stage inference time τ^* and energy E^*.

The main components of DONNA include a profiler and a partitioning algorithm, as shown in Fig. 1b. The profiler analyzes the DNN on each device type[1] to produce a profile with (1) per-layer computation time, (2) per-layer computation energy consumption, and (3) size of each layer's output tensors. The output of the profiler is fed to the partitioning algorithm (see Section II-B for more details). The partitioning algorithm also receives information about the set of available devices $|\mathbb{D}| = D$, communication medium, and the weight coefficient α. Using the dynamic programming minimization technique presented in [9], the algorithm minimizes the cost function to find the slowest inference stage and optimal inference energy where throughput can be estimated from the inference slowest stage. This helps to identify optimal model splits and to choose a subset of devices $\mathbb{S} \subseteq \mathbb{D}$ with $S \leq D$ that participate in distributed inference of the model $\mathbb{M}$. Eventually, DONNA schedules 1 to N independent inference processing stages of a pipeline. Each stage is assigned with its corresponding split from the model to a device from the set $\mathbb{S}$. Energy consumption and throughput can be estimated when all participating devices are identified. The pipeline system is assumed to support asynchronous communication, where computation and communication overlap.

[1]An available device in the system is different than a device type. If, for example, three identical CPUs and one CIM device are accessible, then the system has four available devices and two device types.

B. Algorithm and Cost Functions

$h_t(i, s, u)$ represents the time consumed to process the first i layers of the Neural Network (NN) model using the currently chosen participating set of devices s. The output of the i^{th} layer will be transmitted to the device $u \in \mathbb{D} \backslash s$. Therefore, the algorithm tries to find the optimal time $h_t(L, \mathbb{S}, \phi)$ that is consumed to process all layers of the model $\mathbb{M}$ on participating devices $\mathbb{S}$. While $h_e(i, s, u)$ denotes the energy consumption of processing the first i layers, using the currently chosen participating devices s, then communicating the output of the i^{th} layer to device u. The weight of energy optimization is controlled by α. The total inference energy consumption is calculated by adding the energy cost of processing the considered layers. The proposed cost function is denoted by $h_{t+e}(j, s \cup \{u\}, v)$ with variable weighting factor $\alpha \in [0, 1]$ as follows

$$\begin{aligned} &\min_{j,u,v}[\alpha \cdot h_t^{\text{norm}}(j, s \cup \{u\}, v) + (1-\alpha) \cdot h_e^{\text{norm}}(j, s \cup \{u\}, v)] \\ &\text{s.t.} \quad 0 \leq i < j \leq L \quad \text{and} \quad u, v \in \mathbb{D} \backslash s \end{aligned} \tag{1}$$

We define h_t^{norm} representing the maximum pipeline stage in order to minimize inference's slowest stage as shown in (2). $T_{comp}^{\text{norm}}(\{i \rightarrow j\}, u)$ is the normalized time of computing layers i to j in device u. $T_{\text{comm}}^{\text{norm}}(u, v, l_j^{out})$ is the normalized time to communicate the output of the j^{th} layer from device u to v. h_e^{norm} presented in (3) minimizes the inference energy. $E_{\text{comp}}^{\text{norm}}(l_p, d_p)$ is the normalized energy of the computing layer set of layers l_p on the device d_p. $E_{\text{comm}}^{\text{norm}}(d_p, d_{p+1}, l_p^{out})$ is the normalized energy of communicating the output of l_p from device d_p to d_{p+1}. l_j^{out} is the same as in (2)

$$h_t^{\text{norm}}(j, s \cup \{u\}, v) = \quad max \begin{cases} h_t^{\text{norm}}(i, s, u) \\ T_{\text{comp}}^{\text{norm}}(\{i \rightarrow j\}, u) \\ T_{\text{comm}}^{\text{norm}}(u, v, l_j^{out}) \end{cases} \tag{2}$$

$$\begin{aligned} h_e^{\text{norm}}(j, s \cup \{u\}, v) = &\sum_{p=1}^{|s|+1} E_{comp}^{\text{norm}}(l_p, d_p) + \\ &\sum_{p=1}^{|s|} E_{\text{comm}}^{\text{norm}}(d_p, d_{p+1}, l_p^{out}) + E_{\text{comm}}^{\text{norm}}(u, v, l_j^{out}) \end{aligned} \tag{3}$$

Discrepancies in time and energy scales may lead to biased outcomes. Consequently, we normalize each term relative to its maximal value derived from the collective computation and communication operations of all device types.

DONNA's operation is summarized in Algorithm 1. The algorithm explores different paths in the search space. All paths start with the same initial incomplete pipeline stage $h_{t+e}(0, \phi, device) = 0$ (Line 3). Each path starts forming the pipelining stages by exploring different devices and layers combinations (Lines 5-8), hence, leading to different sub-optimal solutions. The algorithm keeps track only of the best sub-optimal solutions (Line 26). Whenever a better path is found, it is tagged (Line 16) so the optimal solution can be traced back through the path to form the optimal distribution

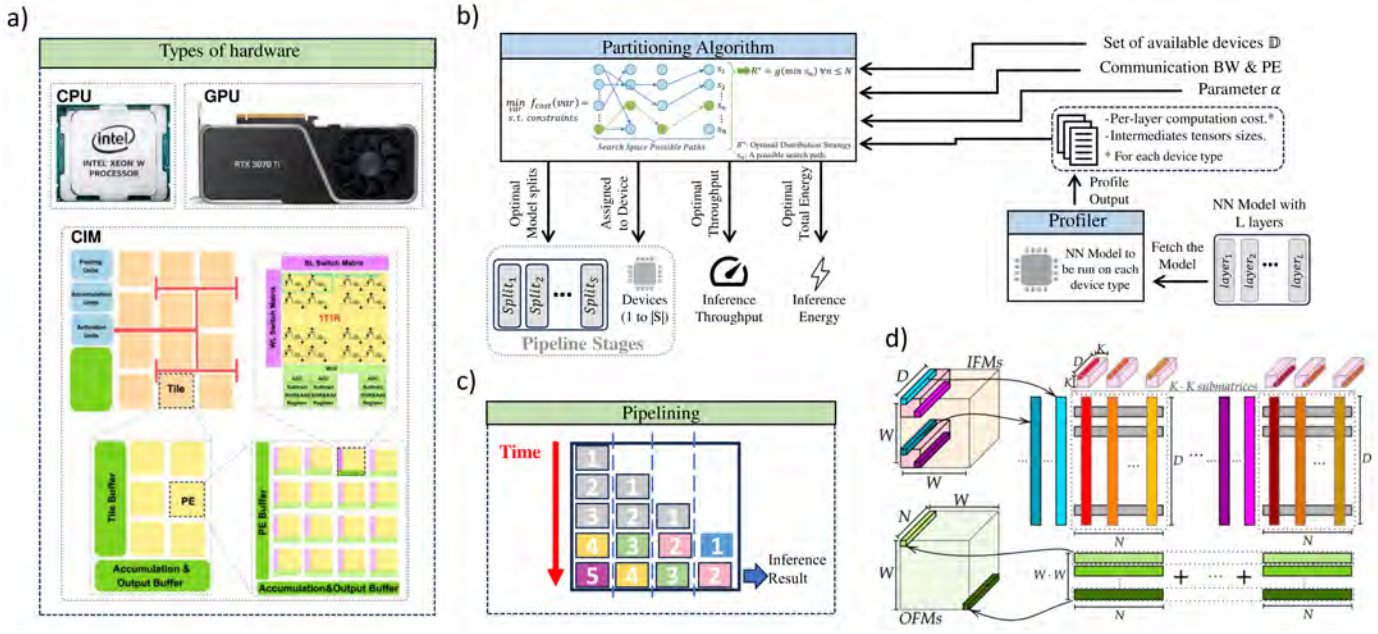


Fig. 1: a) Heterogeneous hardware nodes: 'Intel(R) Xeon(R) W-3323 CPU @ 3.50GHz', 'NVIDIA GeForce RTX 3070 Ti' and CIM-NeuroSim [11]; b) Proposed framework overview: Model Profiling and Partitioning; c) Pipelining stages; d) Convolution layers' kernel mapping used in the CIM simulator [12].

strategy (Lines 36-43). It should be mentioned that this is not a brute-force algorithm as it does not exhaust every possible solution and it excludes some non-optimal ones.

III. Methodology

A. Experimental Setup

In this work, different combinations of computing devices connected via homogeneous communication mediums were tested for ResNet152, VGG19, and VGG8 workloads. Types of computing devices considered include: (1) *NVIDIA GeForce RTX 3070 Ti*, referred to as the GPU device, (2) *Intel(R) Xeon(R) W-3323 CPU @ 3.50GHz*, referred to as the CPU device, and (3) *ReRAM-based CIM accelerator*, referred to as the CIM device). Both the GPU and CPU devices are operated using Linux Ubuntu 20.04.5 LTS. CIM devices are simulated using DNN+NeuroSim [13], which is an end-to-end framework that allows evaluation of chip-level performance and inference accuracy. Neurosim's accuracy was validated and calibrated against a 40nm RRAM-based CIM macro with an error under 1% [11]. Neurosim considers a hierarchical CIM architecture (Fig.1a) consisting of multiple tiles connected via *H-tree* interconnect. Each tile includes processing elements (PE) comprised of a 1T1R crossbar array for synaptic weight implementation. The framework supports methods of weight mapping. 3D convolution kernels are unrolled and mapped into a group of subarrays, maximizing data reuse and reducing energy and latency [12]. As shown in Fig. 1d, the kernel weights are spread with $K \times K$ sub-matrices that are allocated to the Processing Elements (PEs) [12]. Input feature maps' inputs are reused among the PEs as the kernel slides across them. NeuroSim also provides flexible design options at device-level, circuit-level and algorithm-level [13].

TABLE I: Communication Medium Bandwidth and Energy Efficiency

Medium Type	Bandwidth	Efficiency
PCIe-5	64GB/s [15]	6.5pJ/b [16]
High Bandwidth WCC	1GB/s [17]	0.1μJ/b [18]
Low Bandwidth WCC	3.5MB/s [18]	50μJ/b [18]

The first communication medium studied is wired Peripheral Component Interconnect Express (PCIe-5) [14], which is considered in this paper as a cheap communication channel because its communication cost is much less than the computation cost. The other two media are wireless channels, High Bandwidth Wireless communication channel (HB-WCC) and Low Bandwidth Wireless communication channel (LB-WCC), with weaker communication capabilities. Table I summarizes each communication medium's capabilities. Overall, there are 252 scenarios for the four device setup clusters $\{\{4CPUs\}$,$\{2CPUs, 2GPUs\}$, $\{2CPUs, 2CIMs\}$, $\{2CPUs, 1GPU, 1CIM\}\}$ and different α values spanning from 0 to 1 with step of 0.2.

B. Simulation Framework

In this work, we consider the VGG8, VGG19 and ResNet152 models. Due to the CIM simulator (NeuroSim [13]) being restricted to these models, we are not able to support SoTA models such as vision transformers and large language models which we will consider in future work.

Algorithm 1 Proposed DONNA Algorithm Pseudo Code

Require: $\mathbb{D}$: available devices to choose from;
$\mathbb{B}$: bandwidth between devices;
$\mathbb{PE}$: bits transmission power efficiency;
$\mathbb{PF}$: per-layer computation cost and each layer's output size (i.e. P_j) profiles of a model $\mathbb{M}$ with L layers, for each device type.
α: The value of the weighting parameter in (1);
Ensure: T_{opt}: optimal slowest pipeline stage time for optimal throughput Th_{opt};
E_{opt}: optimal total inference energy consumption;
$\mathbb{R}$: optimal distribution strategy;

procedure PARTITION($\mathbb{D}, \mathbb{B}, \mathbb{PE}, \mathbb{PF}, \alpha$) // Start the search algorithm
Initial $h_{t+e}(i,s,u), h_t^{\text{norm}}(i,s,u), h_e^{\text{norm}}(i,s,u) \leftarrow +\infty \ \forall \ i \in L, s \subseteq \mathbb{D}, u \in \mathbb{D}$;
Initial $h_{t+e}(0,\Phi,u), h_t^{\text{norm}}(0,\Phi,u), h_e^{\text{norm}}(0,\Phi,u) \leftarrow 0 \ \forall \ u \in \mathbb{D}$;
Initial $cost^{opt} \leftarrow +\infty$
for $i = 0$ *to* $L-1$ **do**
  for each subset $s \subseteq \mathbb{D}$ **do**
    for each $u \in \mathbb{D} \backslash s$ **do**
      for $j = i+1$ *to* L **do**
        $t_{comp}, e_{comp} = T_{comp}^{norm}(\{i \rightarrow j\}, u), E_{comp}^{norm}(\{i \rightarrow j\}, u)$
        if $j == L$ **then**
          $t_{cost} = max(t_{comp}, h_t^{\text{norm}}(i,s,u))$
          $e_{cost} = e_{comp} + h_e^{\text{norm}}(i,s,u)$
          $C = \alpha \cdot t_{cost} + (1-\alpha) \cdot e_{cost}$
          if $C < cost^{opt}$ **then**
            $cost^{opt} = C$;
            $index = (i,s,u)$
          end if
        else
          for each $v \in \mathbb{D} \backslash s \backslash \{u\}$ **do**
            $t_{comm}, e_{comm} = T_{comm}^{norm}(u,v,P_j), E_{comm}^{norm}(u,v,P_j)$
            $t_{cost} = max(t_{comp}, t_{comm}, h_t^{\text{norm}}(i,s,u))$
            $e_{cost} = e_{comp} + e_{comm} + h_e^{\text{norm}}(i,s,u)$
            $C = \alpha \cdot t_{cost} + (1-\alpha) \cdot e_{cost}$
            if $C < h_{t+e}(j, s \cup \{u\}, v)$ **then**
              $h_{t+e}(j, s \cup \{u\}, v) = C$
              $p(j, s \cup \{u\}, v) = (i,u)$; // Record the precursor
              $h_t^{\text{norm}}(j, s \cup \{u\}, v) = t_{cost}$
              $h_e^{\text{norm}}(j, s \cup \{u\}, v) = e_{cost}$
            end if
          end for
        end if
      end for
    end for
  end for
end for
// Find the optimal strategy where $\mathbb{S} = s \cup \{u\}$
$(i,s,u) = index$;
$Add(i+1 \rightarrow L, u)$ *to* $\mathbb{R}$;
while $i > 0$ **do**
  $(i,u) = p(index)$;
  $Add(i+1 \rightarrow index[0], u)$ *to* $\mathbb{R}$;
  $index = (i, s \backslash u, u)$;
end while
return $Th_{opt}, E_{opt}, \mathbb{R}$ // From $\mathbb{R}$ estimate Th_{opt} and E_{opt}
end procedure

The framework is implemented in PyTorch V1.13. Both the profiler and the partitioning algorithm are implemented in Python3. The partitioning algorithm is based on two pillars: a cost function that the optimal partitioning should minimize and a minimization algorithm. Dynamic programming, using the memoization technique, is used as the minimization algorithm. Its main advantage is that it reduces algorithm computation overhead by calling already calculated sub-problems from memory instead of recomputing them and also by skipping computing large sub-problems that will not be an optimal solution.

1) Performance of CPU and GPU: The Python time() function [19] was used to estimate the per-layer computation times of running the model on the GPU device. Each layer is executed one hundred times, and then the computation time is averaged and taken as the computation time of that layer. The pyJoules library [20] was used to estimate computation time and energy for the CPU device.

The pynvml.smi library [21] was used for the GPU device to estimate the average power of executing a certain model layer, which was then multiplied by its execution time. This is averaged over 100 executions to determine each layer's computation energy consumption. CPU measurements with pyJoules were similarly averaged over 100 runs.

2) Performance of CIM: Computation time and energy estimations in CIM device type are different from GPU and CPU estimations as they come from a simulator and not a physical device. The DNN+NeuroSim [22] simulator was used, with its default parameters, to estimate both per-layer computation time and per-layer computation energy consumption. NeuroSim provides these calculations after defining the intended NN architecture and setting CIM device parameters in the simulator. Table II summarizes how computation cost measurements were taken.

TABLE II: Computation Cost Profiling Tools.

Platform	Time Profiling	Energy Profiling
NVIDIA RTX3070Ti	Python time library [19]	nvidia-smi [21]
Intel Xeon CPU	Python pyJoules library [20]	Python pyJoules library [20]
DNN+NeuroSim CIM	DNN+NeuroSim output [22]	DNN+NeuroSim output [22]

3) Communication cost estimation: The partitioning algorithm uses the device's Bandwidth (BW) and transfer power efficiency to calculate communication time and energy costs where (4) and (5) calculate communication time and communication energy, respectively.

$$T_{comm}(l_j^{out}, BW) = \frac{l_j^{out}}{BW} \tag{4}$$

$$E_{comm}(l_j^{out}, P_{eff}) = l_j^{out} \cdot P_{eff} \tag{5}$$

T_{comm} and E_{comm} are the communication time and energy costs, respectively, to transfer a layer's output of size l_j^{out} through a medium with a transfer rate of BW and a power efficiency of P_{eff} in pJ/s.

4) Inference energy consumption and throughput estimation: After assigning each model split to its corresponding device type, the framework utilizes device types' profiles to estimate the inference energy and throughput. The computation energy of each assigned split and communication energy resulting from this distribution strategy are summed together to form the total inference energy consumption. The framework also finds the slowest pipelining stage of computation and communication times to take its inverse as the inference throughput.

IV. Results and Discussion

A. Models Profiling Results

Fig. 2a–b present per-layer computation profiles of workloads running on compute nodes described in Section III-A. In general, the GPU is the fastest in terms of computation speed, and the CIM device is the most efficient compute device in terms of energy consumption. Also, it can be seen that the

CIM device is the slowest in computations, except for the last one or two layers of each model. This is a result of how the CIM technology is designed. The CIM design is optimized for matrix-vector multiplications which suits more FC layers than convolution layers.

B. Models Partitioning Results

1) Simultaneous Energy and Throughput Optimization: Fig. 2c–k show possible ResNet152, VGG19, and VGG8 partitioning scenarios across different combinations of devices, communication medium, and variable α. The marker shape in the plot represents the set $\mathbb{D}$, whereas its color describes α's value. It is important to note that not all available devices $\mathbb{D}$ may participate in distributing the inference (i.e., $|\mathbb{S}| \leq |\mathbb{D}|$). The horizontal subplots describe the effect of each communication medium on the algorithm's behavior. The algorithm tries to minimize inference energy (i.e., $\alpha \to 0$) or maximize inference throughput (i.e., $\alpha \to 1$). When $\alpha = 0$, the algorithm assigns all model layers, not shown on the plot, to the most energy-efficient device. Although this minimizes the total energy, it may harm the performance by preventing other devices from participating, leading to slower throughput as the system is not utilizing the speedup gained from pipelining. On the other hand, DONNA maximizes inference throughput by trying to enable the fastest devices to participate and does not consider energy consumption when $\alpha = 1$. This will always ensure maximized throughput. Nevertheless, tuning α to reduce energy with negligible effect on the throughput is possible. For example, while finding ResNet152 optimal split on HB-WCC and $\mathbb{D} = \{4CPUs\}$, varying α from 1 to 0.8 reduced energy by 7.6% with negligible effect (less than 0.5%) on throughput.

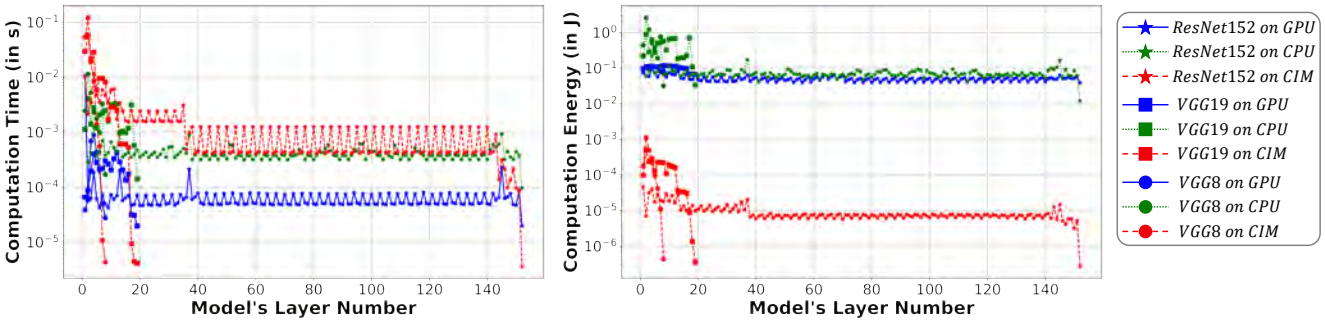
Simultaneous optimization of throughput and energy consumption (i.e. $0 < \alpha < 1$) greatly depends on computation cost compared to the communication cost. We used the same device configurations for all simulations and swept α from 0 to 1 for three different communication mediums. Energy reduction is negligible in the three models when PCIe-5 was used in distributing homogeneous devices as shown in Fig. 2c, f and i. As the communication cost is very cheap compared to the computation cost, reducing energy by reducing α encourages fewer devices to participate, preventing the algorithm from exploiting the advantage of pipelining speedup and the saved communication energy is small compared to the computation energy. Furthermore, more expensive communication media result in overlapping optimization points. For example, the algorithm chose a single GPU to perform the inference on both $\mathbb{D} = \{2CPUs, 2GPUs\}$ and $\mathbb{D} = \{2CPUs, 1GPU, 1CIM\}$ when $\alpha = 1$ as shown in Fig.2e resulting in two overlapping optimization points.

The number and diversity of available devices in a set $\mathbb{D}$ play an important role. Optimization points are more spread across the energy-throughput space tending towards the optimal performance region when $\mathbb{D} = \{2CPUs, 1GPU, 1CIM\}$ mainly due to the efficient performance of CIM and the computational performance of the GPU. In other words, the higher the compute diversity is, the more flexibility is offered to obtain the intended performance. This advantage is most apparent in Fig.2c. Furthermore, comparing Fig. 2i, j and k to the rest of the subplots, the number of model layers also impacts inference distribution. A smaller number of layers limits the spread of optimization points across the throughput-energy space.

2) Optimized Distributed Inference vs. Single Device Performance: Although the GPU used provides the best trade-off compared to the other device types and the NN models in our work can fit into a single GPU, distributing them over multiple devices can provide higher performance. Fig. 3 shows the trade-off of distributing ResNet152 with the normalized scales for inference speedups, total energy consumption, and average device energy consumption. Here, the average device energy consumption is computed as the total inference energy over all participating devices, hence, it is different from the per-device energy of individual participating devices. Therefore, some scenarios show an improved performance in either throughput or energy consumption, while others show worse performance compared to the baseline, which, in our case, is the performance of a single GPU. For instance, Scenarios 4 and 10 with $\mathbb{D} = \{2GPUs, 0CIMs, 2CPUs\}$ and $\mathbb{S} = \{2GPUs, 0CIMs, 2CPUs\}$ achieve faster throughput than a single GPU, showing the benefit of inference distribution using our proposed framework. The relatively cheap communication medium HB-WCC enabled more devices to participate with minimal communication energy overhead, yielding an increased throughput speedup with a slight increase in total energy consumption. In addition, this distribution resulted in a lower average device energy consumption compared to that of a single GPU. The proposed framework in Scenario 10 could speed up the estimated throughput 4.26× and achieve a 3.6× per-device energy reduction. On the other hand, using a costly channel such as LB-WCC (Scenario 16) does not yield a noticeable speedup and results in higher total energy consumption. Nevertheless, the average per-device energy consumption is less than the computation energy of a single GPU.

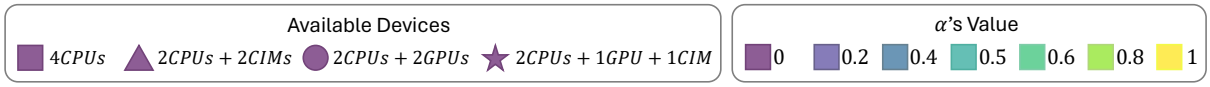
Furthermore, augmenting the system with CIM allows energy-efficient solutions designing with a trade-off in throughput. Scenarios 2 and 8 show how much energy can be saved when inference is distributed on CIM devices. However, inference throughput dropped to less than half of a single GPU throughput due to CIM device design limitations mentioned at the beginning of this section. Using only CIM devices on a costly channel such as LB-WCC (i.e., Scenario 14) led to a drop in throughput and a significant increase in communication energy consumption due to the slowness of CIM device compared to GPUs in performing convolutions that represent more than 90% of the used DNN models and the high communication energy of LB-WCC.

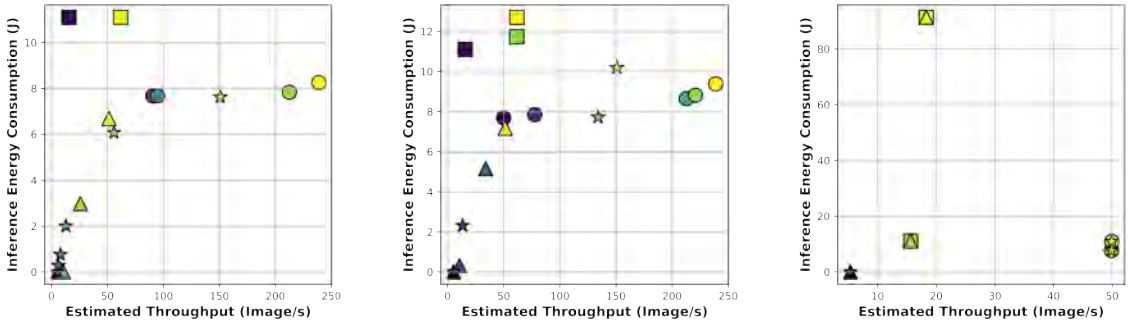
3) CIM as an Edge-Device: A real-world scenario would consist of multiple devices at the edge that can communicate with the cloud. Unlike previous experimentation, this scenario does not assume all devices communicate with the same medium. Edge devices communicate with each other



(a) Time profile

(b) Energy profile

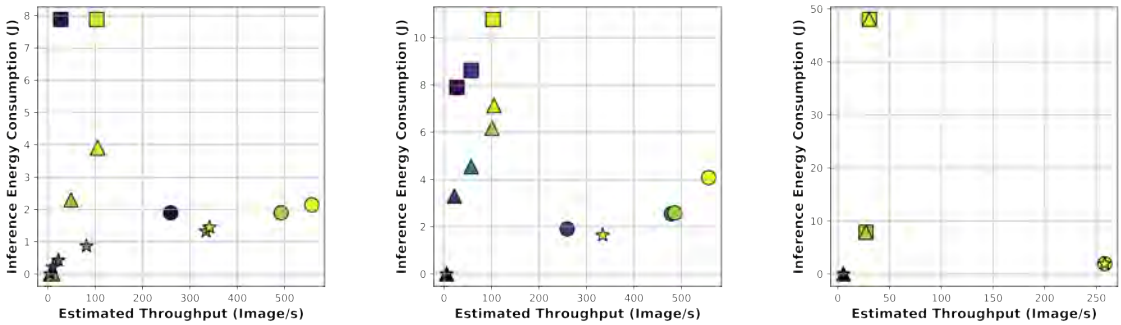




(c) ResNet152 on PCIe-5

(d) ResNet152 on HB-WCC

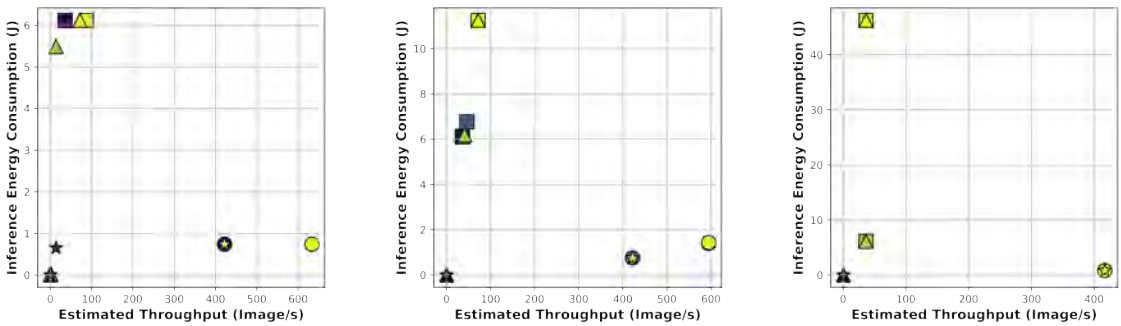
(e) ResNet152 on LB-WCC



(f) VGG19 on PCIe-5

(g) VGG19 on HB-WCC

(h) VGG19 on LB-WCC



(i) VGG8 on PCIe-5

(j) VGG8 on HB-WCC

(k) VGG8 on LB-WCC

Fig. 2: (a-b) Computation costs profiles for different DL workloads; (c-k) Distribution performance using DONNA on different communication channels, available devices setup, and different α values when partitioning ResNet152, VGG19, and VGG8 on the communication mediums.

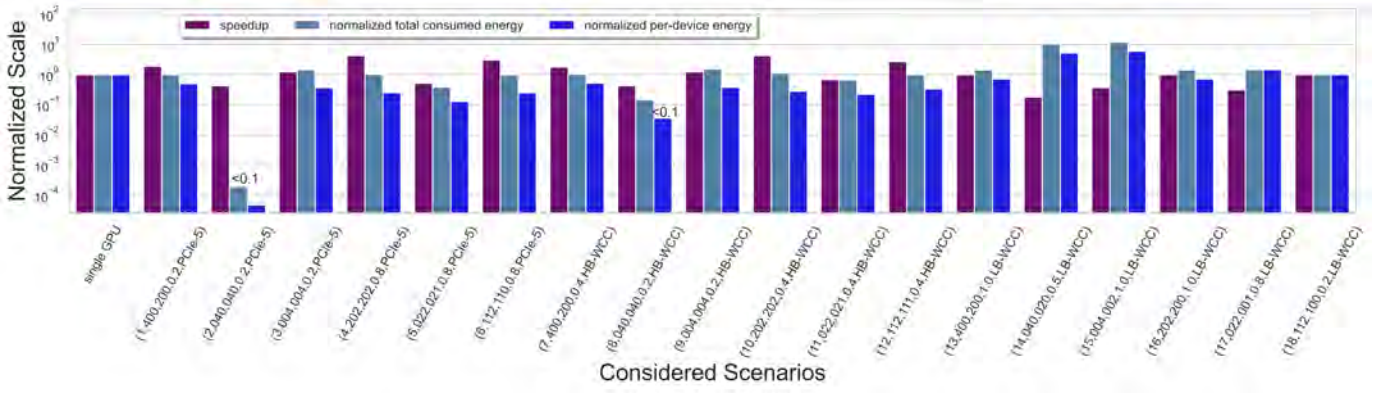
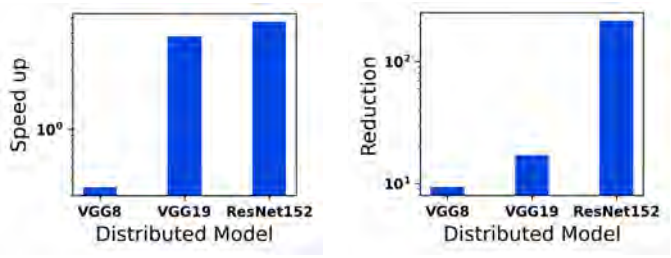


Fig. 3: Inference throughput speedup, total energy consumption and per-device energy consumption when distributing ResNet152: single GPU is a baseline scenario and Scenarios (x, $y \in \mathbb{D}$, $z \in \mathbb{S}$, α, communication medium), where x is the scenario number; device combinations y and z correspond to number of available and participating GPUs, CIMs and CPUs, respectively.



(a) Throughput Speedup (b) Energy Reduction

Fig. 4: Comparing distributed inference on CIM devices to offloading all computations to a GPU cloud. (a) shows throughput speedup gains. (b) shows energy reduction gains.

through the same high data rate channel (i.e. HB-WCC) while communication to the cloud is modeled with a slower network (i.e. LB-WCC) under the premise of heavy traffic routed to the cloud. A single GPU or CPU is used as the cloud and CIM devices are used as the edge devices for their very efficient energy consumption which is a crucial metric at the edge. Although DONNA tried to only maximize throughput (i.e. $\alpha = 1$), it allocates all layers to the edge devices and does not offload any computations to the cloud whether it is a GPU or a CPU.

This suggests that in terms of both energy and throughput, the cost of distributing the inference among edge devices is less than that of partially offloading computations to the cloud. This can be seen in Fig. 4 where offloading all computations to the cloud (i.e. GPU) is the baseline. It can be noticed from the figure that the larger the workload is, the more performance gains are achieved in such a setup. Energy reduction is always achieved and can reach more than $\times 200$ for the largest workload which shows a core advantage of CIM device's energy efficiency. It should be noted that the algorithm does not instruct the edge device to offload all VGG8 computations to the cloud as it is forced to process at least the first layer on the edge device, hence, transmitting raw data to the cloud feeding VGG8 achieves higher throughput as shown in Fig.4a.

In general, increasing α encourages the algorithm to find a split that will lead to higher throughput and possibly increase inference energy consumption and vice versa. Decreasing α in homogeneous compute nodes and very cheap communication channels may lead to a limited number of participating devices, a significant throughput decrease with negligible energy reduction as communication overhead is already small. Furthermore, NN models with more layers allow more flexibility in finding the intended optimized performance. Finally, the more diverse compute nodes available to choose from, the smoother the set of solutions found by varying α.

V. Conclusion

The DONNA framework concurrently optimizes distributed inference for both throughput and energy consumption, a key dual objective. Experimental evaluation was conducted on ResNet152, VGG19, and VGG8 architectures leveraging the ImageNet dataset for heterogeneous systems consisting of GPU, CPU, and CIM. Compared to the performance of a single GPU, DONNA can achieve about $4\times$ speedup gained from pipeline processing with around $4\times$ per-device energy reduction. This improvement was realized by judiciously allocating layers to devices with optimal energy efficiency within a heterogeneous configuration. The algorithm finds a smooth Pareto-like curve on the throughput-energy space when CIM devices participate thanks to having devices with diverse performance capabilities, allowing the algorithm to explore a broader spectrum of optimization points. Our future work includes scaling the framework to support large language models, exploring other model-splitting techniques, using near-exact computation profiling and developing a more accurate communication model.

VI. Acknowledgement

This work has been partially supported by King Abdullah University of Science and Technology CRG program under grant number: URF/1/4704-01-01

REFERENCES

[1] M. Dehghani, J. Djolonga, B. Mustafa, P. Padlewski, J. Heek, J. Gilmer, A. Steiner, M. Caron, R. Geirhos, I. Alabdulmohsin *et al.*, "Scaling vision transformers to 22 billion parameters," *arXiv preprint arXiv:2302.05442*, 2023.

[2] T. Brown, B. Mann, N. Ryder, M. Subbiah, J. D. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell *et al.*, "Language models are few-shot learners," *Advances in Neural Information Processing Systems*, vol. 33, pp. 1877–1901, 2020.

[3] "Nvidia h100 tensor core gpu datasheet," https://resources.nvidia.com/en-us-tensor-core/nvidia-tensor-core-gpu-datasheet, accessed: 2023-06-13.

[4] "Ai memory wall," https://medium.com/riselab/ai-and-memory-wall-2cb4265cb0b8, accessed: 2023-06-12.

[5] F. Staudigl, F. Merchant, and R. Leupers, "A survey of neuromorphic computing-in-memory: architectures, simulators, and security," *IEEE Design & Test*, vol. 39, no. 2, pp. 90–99, 2021.

[6] D. Zhao, Y. Wang, J. Shao, Y. Chen, Z. Guo, C. Pan, G. Dong, M. Zhou, F. Wu, W. Wang *et al.*, "Compute-in-memory for numerical computations," *Micromachines*, vol. 13, no. 5, p. 731, 2022.

[7] K. Smagulova, M. E. Fouda, F. Kurdahi, K. N. Salama, and A. Eltawil, "Resistive neural hardware accelerators," *Proceedings of the IEEE*, vol. 111, no. 5, pp. 500–527, 2023.

[8] X. Hou, Y. Guan, T. Han, and N. Zhang, "Distredge: Speeding up convolutional neural network inference on distributed edge devices," in *IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, 2022, pp. 1097–1107.

[9] Y. Hu, C. Imes, X. Zhao, S. Kundu, P. A. Beerel, S. P. Crago, and J. P. Walters, "Pipeedge: Pipeline parallelism for large-scale model inference on heterogeneous edge devices," in *Euromicro Conference on Digital System Design (DSD)*, 2022, pp. 298–307.

[10] Y. Kang, J. Hauswald, C. Gao, A. Rovinski, T. Mudge, J. Mars, and L. Tang, "Neurosurgeon: Collaborative intelligence between the cloud and mobile edge," *ACM SIGARCH Computer Architecture News*, vol. 45, no. 1, pp. 615–629, 2017.

[11] A. Lu, X. Peng, W. Li, H. Jiang, and S. Yu, "NeuroSim validation with 40nm RRAM compute-in-memory macro," in *IEEE International Conference on Artificial Intelligence Circuits and Systems (AICAS)*, 2021, pp. 1–4.

[12] X. Peng, R. Liu, and S. Yu, "Optimizing weight mapping and data flow for convolutional neural networks on processing-in-memory architectures," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 67, no. 4, pp. 1333–1343, 2019.

[13] X. Peng, S. Huang, Y. Luo, X. Sun, and S. Yu, "DNN+NeuroSim: An end-to-end benchmarking framework for compute-in-memory accelerators with versatile device technologies," in *2019 IEEE international electron devices meeting (IEDM)*. IEEE, 2019, pp. 32–5.

[14] "Doubling bandwidth in under two years: PCI Express base specification revision 5.0, version 0.9 is now available to members," https://pcisig.com/doubling-bandwidth-under-two-years-pci-express\%C2\%AE-base-specification-revision-50-version-09-now, accessed: 2023-06-13.

[15] "Trenton Systems blog: What is PCIe 5.0?" https://www.trentonsystems.com/blog/what-is-pcie-5.0, accessed: 2023-06-13.

[16] "Nvidia grace-hopper white paper," https://resources.nvidia.com/en-us-grace-cpu/nvidia-grace-hopper, accessed: 2023-06-25.

[17] "Qualcomm: Everything you need to know about 5g," https://www.qualcomm.com/5g/what-is-5g#:~:text=5G\%20can\%20be\%20significantly\%20faster,(Mbps)\%20average\%20data\%20rates., accessed: 2023-06-20.

[18] E. Björnson and E. G. Larsson, "How energy-efficient can a wireless communication system become?" in *Asilomar Conference on Signals, Systems, and Computers*, 2018, pp. 1252–1256.

[19] "Python library: time — time access and conversions," https://docs.python.org/3/library/time.html, accessed: 2023-06-13.

[20] "pyjoules's documentation," https://pyjoules.readthedocs.io/en/latest/, accessed: 2023-06-13.

[21] "Python bindings to the nvidia management library," https://pypi.org/project/pynvml/, accessed: 2023-06-13.

[22] "DNN+NeuroSim v1.3 GitHub page," https://github.com/neurosim/DNN_NeuroSim_V1.3, accessed: 2023-06-13.