MEDEA: A Multi-objective Evolutionary Approach to DNN Hardware Mapping

Enrico Russo*, Maurizio Palesi*, Salvatore Monteleone†, Davide Patti*, Giuseppe Ascia* and Vincenzo Catania*
*Department of Electrical, Electronic, and Computer Engineering (DIEEI), University of Catania, I-95125 Catania, Italy.
enrico.russo7@studium.unict.it, {maurizio.palesi, davide.patti, giuseppe.ascia, vincenzo.catania}@dieei.unict.it
†Department of Engineering, Niccoló Cusano University, I-00166 Rome, Italy.
salvatore.monteleone@unicusano.it

***Abstract*—Deep Neural Networks (DNNs) embedded domain-specific accelerators enable inference on resource-constrained devices. Making optimal design choices and efficiently scheduling neural network algorithms on these specialized architectures is challenging. Many choices can be made to schedule computation spatially and temporally on the accelerator. Each choice influences the access pattern to the buffers of the architectural hierarchy, affecting the energy and latency of the inference. Each mapping also requires specific buffer capacities and a number of spatial components instances that translate in different chip area occupation. The space of possible combinations, the mapping space, is so large that automatic tools are needed for its rapid exploration and simulation. This work presents MEDEA, an open-source multi-objective evolutionary algorithm based approach to DNNs accelerator mapping space exploration. MEDEA leverages the Timeloop analytical cost model. Differently from the other schedulers that optimize towards a single objective, MEDEA allows deriving the Pareto set of mappings to optimize towards multiple, sometimes conflicting, objectives simultaneously. We found that solutions found by MEDEA dominates in most cases those found by state-of-the-art mappers.**

***Index Terms*—neural network, accelerator, compiler, scheduling, genetic algorithm, multi-objective optimization**

I. Introduction

Domain specific architectures are foreseen as the most viable solution to deal with the end Moore's law and the end of Dennard scaling as well as a departure from general-purpose architectures suffering the unsustainable "Turing tariff" [1]. Deep learning (DL) in general and in particular Deep Neural Networks (DNNs) have shown to be able to solve, sometimes with superhuman accuracy, complex real-world problems in a wide range of applications, including, image recognition, object detection, language translation, and autonomous driving. Thus, in the last few years we have assisted to a proliferation of DSAs tailored to improve the efficiency of DNN computation.

A DNN layer mapping defines temporal and spatial scheduling of multiply-accumulate operations. The design of an efficient DNN hardware accelerator and the selection of the most appropriate mapping for a given DNN model involve the exploration of the ocean of design alternatives (a.k.a. system instances) that cannot be exhaustively searched. The ultimate goal while designing a DNN accelerator is the optimization of different and sometimes conflicting objectives like performance, power, energy, and area. For example, to improve performance we might increase the number of PEs. Doing so, the power and area increase but, as the inference time reduces, it might bring to energy saving.

Current mapping space exploration techniques proposed in literature [2]–[6] are mono-objective in their nature. That is, they try to optimize a single objective (*e.g.*, delay and energy) or a single aggregation function like a linear combination of the objectives or their product (*e.g.*, energy delay product). Thus they provide a single solution that, even if optimal for a specific objective or a specific aggregation of objectives, might not be acceptable for some of the remaining objectives.

As in practical cases, there is not a single system instance that satisfies all the design objectives in a whole, a more appropriate approach is to derive the Pareto optimal set of system instances. By treating optimization objectives separately from each other, *i.e.*, performing multi-objective optimization, allows finding solutions that cannot be found by aggregating the objectives by a single function. For instance, aggregating the objectives by a weighted sum brings to non uniform distribution of the optimal solutions, and more seriously, optimal solutions in non-convex regions are not detected.

In this paper, we advocate the use of multi-objective optimization as the appropriate way to deal with the design space exploration of DNN hardware accelerators. To the best of our knowledge, this is the first work that explores the huge design and mapping space of DNN accelerators in a multi-objective fashion. We use a MOEA based approach which is augmented with a set of domain-specific evolutionary operators with the aim of deriving an approximation of the Pareto optimal set in an efficient way. We use MEDEA to find the approximate Pareto optimal set of system instances that optimize the trade-off inference energy *vs*. inference latency *vs*. area of the accelerator. We compare MEDEA with state-of-the-art schedulers, including, Timeloop/Accelergy [2], [7] and CoSA [3]. We show that, in contrast with the single solution found by the other approaches, the set of solutions found by MEDEA provides the designer a spectrum of alternatives covering several different trade-offs among the design objectives.

II. Related Works

To address the scheduling problem, several frameworks have been proposed. Timeloop/Accelergy [2], [7] includes a mapper

978-3-9819263-6-1/DATE22/©2022 EDAA

which allows generating different mappings. It makes use of an exhaustive and random approach, which takes a long time to find an optimal mapping. MindMappings [4] makes use of a surrogate model to speed up simulation time. Mappings are tested on the trained surrogate model before doing heavy-weight simulation and this allows speeding up search time. The drawback of this approach is that training the surrogate model is necessary. LOMA [5] leverages loop permutation generation and analysis in order to heuristically allocate more data in lower buffers in the architectural hierarchy and minimize costly accesses to higher-level ones. CoSA [3] leverages Mixed-Integer Programming in order to deterministically generate the optimal mapping based on linear constraints without requiring iterative simulations. GAMMA [6], similarly to our approach, makes use of a genetic algorithm leveraging MAESTRO [8] cost model. In the above works, the mapping space for a given architecture of the accelerator is explored. With MEDEA architectural parameters, including, the number of processing elements and the buffer size, are explored as well thus largely increase the design space. ZigZag [9], a framework for the joint exploration of the DNN architecture and mapping space has been recently proposed. However, like in the previous works, the exploration is limited to the optimization of a single objective. MEDEA explores the design space in a multi-objective fashion thus providing a set of trade-off design solutions among which the user will select the most appropriate one to her/his specific requirements.

III. Background

A. DNNs Accelerators

Deep Neural Networks Accelerators are domain-specific hardware architectures specifically designed to execute neural networks algorithms more efficiently than general-purpose architectures [10]. The resource thrift of accelerators comes from their ability to exploit data reuse. The basic principle is that, in a memory hierarchy, accessing higher levels (*e.g.*, DRAM) involves a much higher energy cost than accessing lower ones, (*e.g.*, registers). Thus, it is essential to find the computation scheduling that makes it possible to keep and reuse the same data as much as possible in lower level buffer before accessing higher levels to retrieve new data. DNNs accelerators also exploit parallelization to reduce latencies. Many architectures contain an array of Processing Elements (PEs) [11]. Each PE contains some Multiply-Accumulate Units (MACs). PEs are interconnected through a specialized Network-on-Chip (NoC) and, in order to avoid high communication costs needed to transfer data from buffers to PEs, it is possible to exploit spatial data reuse. In fact, with proper scheduling, the same data can be sent to multiple PEs with a multicast transmission [12].

B. Loop Nest Notation and Mapping

Tensor operations can be described algorithmically using the loop nest notation. Fig. 1 shows the loop nest representation of a dot product between the two arrays A and B of length 4. Consider a very simple architecture with a shared buffer feeding two MAC units. In order to achieve the lowest latency

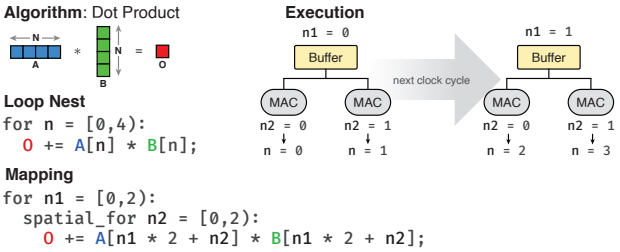


Fig. 1: Loop nest notation representing dot product spatial mapping on a simple hardware architecture.

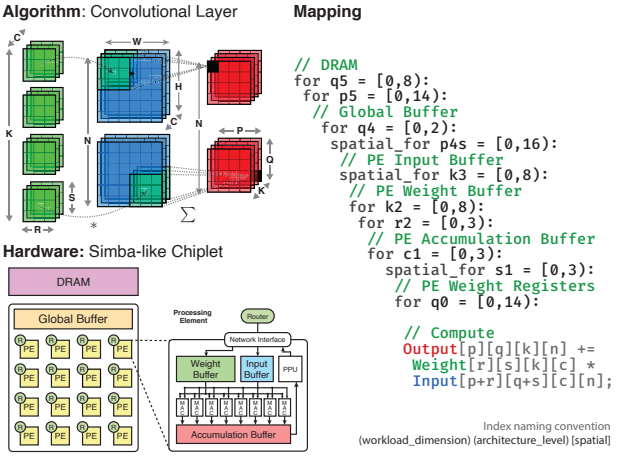


Fig. 2: Mapping loop nest example of a convolutional layer with $N = 1$, $R = S = 3$, $P = Q = 224$, $C = 3$ and $K = 64$

possible, a proper mapping has to be chosen. In the Figure, we show a mapping that leverages loop tiling and parallelization to distribute the calculation among the MAC units.

In Convolutional Neural Networks (CNNs), most of the layers are convolutional and fully connected layers. Convolutional layers can be described as a 7D loop nest, as shown in Fig. 2. With more than one dimension, there is also the possibility to change the order of loops in the loop nest. Different loop orderings generate different data movement patterns called *dataflows*. Scheduling such a loop nest on a complex architectural hierarchy is challenging. A mapping is the result of four choices at each architectural layer as follows:

- **Loop Tiling** (also called Index Factorization) consists in choosing loop bounds for tiled loops. Choosing proper bounds is important to make data fit in buffers.
- **Loop Permutation** (also called Loop Ordering) consists in choosing which loops will be outer loops and which inner ones. The index of the outer loops will change less often than that of the inner ones. This can induce stationarity for some elements limiting their impact on memory access energy.
- **Spatial Mapping** (also called Parallelisation). Spatial loops describe how the workload is distributed on different hardware instances. If the hardware instances are

organized in a grid fashion, such as PEs in many accelerators, spatial loops can be relative to the horizontal or vertical direction. Spatial loops induce different communication patterns in interconnection networks.

- **Datatype Bypass**. A datatype is a tensor of the workload. In a convolutional layer, the datatypes involved are Outputs, Inputs, and Weights. Datatype Bypass means choosing which datatype will be stored in a particular buffer and which will be bypassed.

C. Multi-Objective Genetic Algorithms

Genetic Algorithms (GAs) are stochastic optimization algorithms inspired by natural selection and natural genetics [13]. These algorithms work on a population of individuals or chromosomes, simulating evolution mechanisms. The fundamental parts of these algorithms are reproduction and survival. In the former, two individuals of the population are selected, combined, and mutated to generate two offsprings. In the latter, the best individuals are selected to be part of the next generation population. These processes are repeated until termination criteria are met.

In single-objective GAs, the survival phase is based on a single metric called fitness function. Their aim is to find the solution having the best fitness. The fitness function can aggregate multiple objectives. However, when the objectives are conflicting each other it is desirable to derive not a single solution but a set of non-dominated, *i.e.*, Pareto optimal solutions.

In a minimization problem involving multiple cost functions, we say that the individual x Pareto dominates the individual y, and we write $x \prec y$, if the following conditions are met:

$$\forall i \in \{1, \ldots, K\} : o_i(x) \leq o_i(y)$$
$$\exists j \in \{1, \ldots, K\} : o_j(x) < o_j(y),$$

here K is the number of objective functions $o_i(\cdot)$. When a solution is not dominated by any other individual, it is said to be Pareto optimal. The set of Pareto optimal solutions is called Pareto set. For a solution in the Pareto set, it is not possible to improve one of the objectives without worsening another. The goal of multi-objective GAs is to approximate the Pareto set of the solution space [14].

IV. MEDEA

MEDEA is based on the multi-objective evolutionary algorithm NSGA-II [15]. After an initialization phase in which an initial population of individuals (*i.e.*, mappings) is generated randomly, the algorithm main loop is executed until a configurable number of generations has been simulated. As shown in Fig. 3, in the main loop, individuals are selected in pairs from the parent population for crossover. Each pair generates two offspring individuals that undergo mutation. Each individual in the offspring population is first evaluated using Timeloop Model on the reference architecture in order to obtain information about buffer occupancy and spatial instances utilization. Then, the minimal architecture, that is that with the minimum resources needed by the mapping, is built. The Timeloop/Accelergy framework is again queried on this new architecture and energy, latency and area objectives values are assigned to the individual. The parent and offspring populations are then merged. Fast non-dominated sorting is applied and individuals are grouped into fronts of increasing rank based on their objective values. Individuals in the first front are those belonging to the Pareto front of the merged population. Subsequent fronts are constructed removing fronts with lower rank from the merged population and then calculating the Pareto front. The rank is taken into account in the survival phase to generate the new parent population. Individuals of the lowest rank are transferred to the new generation. In the event that a front is too large to be transferred entirely to the next generation, the crowding distance, *i.e.*, how much each individual is different from adjacent ones on the front to which it belongs in terms of objective metrics, is taken into account.

A set of problem-specific genetic operators have been designed with the goal of improving the efficiency of the MOEA implemented in MEDEA. The specialized crossover, mutation, and a novel parallelizing mutation operator are discussed in the rest of this section.

A. Crossover

The idea behind the crossover operator is that it is possible to combine part of the genomes of two individuals to possibly get a better offspring. This is possible if the "best" sections of the respective genomes are combined together. In our case, we imagine, for example, that two mappings can generate the same inference latency metric for different reasons. For instance, in architecture with 16 PEs, each composed of 16 MACs, it is theoretically possible to achieve the same performance either by using only one MAC in each PE or every MAC unit in only one PE. These two possibilities are expressed in the loop nest notation by using a spatial loop in two possible levels of the architecture hierarchy.

The crossover operator in MEDEA swaps the mapping strategy at the same architectural level between two mappings to achieve a better combination. If the parent mappings respect permutation constraints at the swapped level, also the offspring will respect these constraints. The drawback of this approach is that the overall mapping could not be valid anymore. This is because the mapping has to respect the workload dimensions, this means that the product of loop tiling bounds relative to a specific dimension has to be equal to the dimension size of the workload. To address this issue, factor compensation is performed after the crossover operation. If after the crossover a factor has been increased, we search for another loop relative to the same dimension in the mapping loop nest and reduce its bound in order to keep the total product of bounds always the same. We start from the DRAM level because it is heuristically better to reduce accesses to the top levels of the architecture.

When the factor is decreased we compensate in DRAM as well because it is the memory with the largest capacity, so it is unlikely that the offspring mapping will be invalidated because of non respected memory capacities.

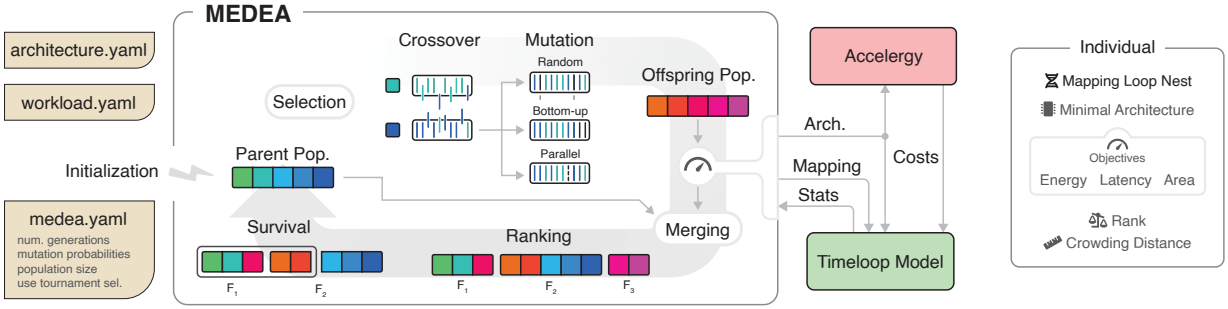


Fig. 3: MEDEA Overview

B. Mutation

MEDEA implements three mutation operators:

1) Random Mutation: This operator widely used in genetic algorithms makes parts of the genome vary randomly. In our case, this operation is carried out by exchanging the order of two random loops within the same architectural level and exchanging two loops afferent to the same workload dimension between two architectural levels. Due to its random nature, the output of this operator may not respect the capacities of the buffers, so there are no guarantees of validity. Offsprings that are no longer valid when mutated are replaced by valid individuals generated randomly.

2) Bottom-up Fill Mutation: Accesses to higher memory incur in high energy penalties. For this reason, good scheduling should allocate more data in lower-level buffers. This facilitates data reuse and reduces costly access to off-chip memory. The occupied capacity of a buffer for each tensor (dataspace) can be calculated by the following expression:

$$\prod_{d=1}^{D}\prod_{l=0}^{L} f_{d,l} \tag{1}$$

where L is the rank of the buffer in the hierarchy (starting from the lower levels), D is the number of dimensions of the considered workload and $f_{d,l}$ is the loop bound at level l relative to the dimension d.

By the bottom-up fill mutation operator introduced in MEDEA a level is chosen randomly using an exponential distribution, so that the lower levels of the memory hierarchy are more likely to get filled. Then, the current occupied capacity of the current level is taken into account and the loop bound mutation of a random dimension, needed to increase buffer occupancy, is calculated inverting (1). If suitable dimension and factor are found, the loop is updated and factor compensation is performed to keep the mapping valid.

3) Parallelizing Mutation: The key for latency reduction is parallelization. In MEDEA the parallelizing mutation takes into account architecture specifications in order to accurately parallelize computation on the available spatial instances. For each architectural level and for each spatial dimension (*e.g.*, x and y in a mesh-based organization of PEs), a spatial loop is either updated or created considering the current parallellization factor which arise from the product of current spatial loops bounds and the maximum possible fanout derived from architecture specifications.

TABLE I: Simba-like simulated microarchitecture parameters.

DRAM		Chiplet		PE	
Technology	LPDDR4	Technology	45nm	Clock Frequency	1 GHz
Bandwidth	17.9 GB/s	Number of PEs	16	Weight Buffer Size	32 KiB
		Global Buffer Size	64KiB	Input Buffer Size	8 KiB
				Accum. Buffer Size	3 KiB
				MACs per PE	64

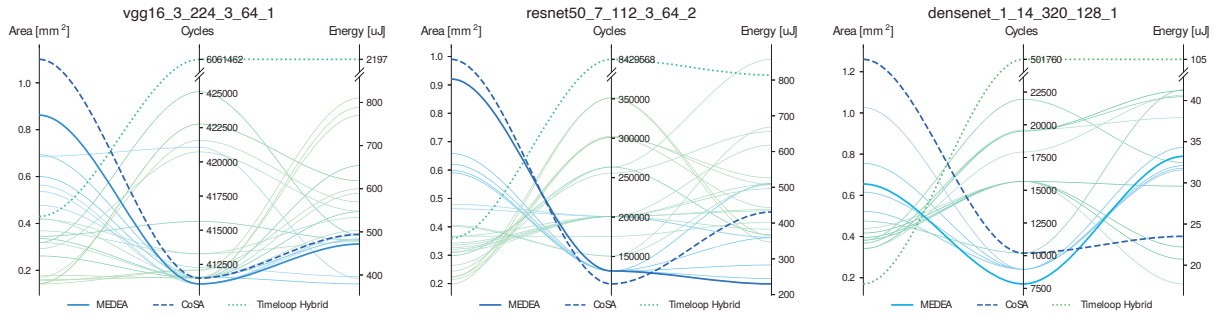
V. Evaluation

In this section, we assess MEDEA and compare it with the hybrid search algorithm, used by the Timeloop mapper [2], and CoSA [3] a state-of-the-art approach for scheduling DNN accelerators that uses a constrained-optimization-based approach. We used LAMBDA simulation platform [16] and we considered a Simba-like [17] chiplet as the reference architecture. Table I lists the main characteristics of the considered architecture. Bit precisions adopted are 8 bits for weights and inputs tensors and 24 bits for partial sums. It should be noted that the size of the buffers and the number of spatial instances specified in MEDEA input files are the upper bounds within which a mapping will be considered valid. In the evaluation phase of each mapping, the reference architecture is replaced with the minimal architecture in order to calculate the area. To compare the area obtained with other approaches, we also applied this process to their solutions.

For the evaluations, we configured MEDEA to simulate 15 generations of 120 individuals with 40% Random Mutation probability, 70% Bottom-up Fill Mutation probability, and 70% Parallel Mutation probability. In simulations involving the Timeloop Mapper, we considered the Hybrid search algorithm that algorithm randomly samples the index factorization space, *i.e.*, the space of all the possible loop bounds assignments, and then visits each possible loop permutation for each assignment. We configured 500 subsequent suboptimal mappings visited as victory conditions and ran the mapper on 6 threads.

A. Per-layer Pareto Optimal Solutions

MEDEA schedules a neural network layer and provides a set of Pareto optimal solutions rather than a single solution



(a) (b) (c)

Fig. 4: Parallel coordinate plot of Pareto optimal mappings found by MEDEA for different workloads compared to single solutions found by CoSA and Timeloop Mapper. Only best 30 mappings in terms of latency are shown.

like the other approaches that optimize a single objective. Fig. 4 shows the parallel coordinate plot of the Pareto efficient mappings found by MEDEA for three different layers, each one belonging to a different network. For readability reasons, we show only the 30 lowest latency mappings. The plot titles adopt the naming convention $neuralnet\_R\_P\_C\_K\_stride$, assuming $R = S$ and $P = Q$. The MEDEA solution with minimum latency is highlighted. We also compare MEDEA's solutions with the ones found by CoSA and Timeloop Hybrid for the same workload and architecture.

From Fig. 4a it can be seen that the CoSA solution is Pareto dominated by one of the MEDEA solutions. We can also observe that two solutions exist with better area and energy at the cost of little increase in latency. When energy consumption is the priority, such mappings can be chosen by the designer who intends to accept the trade-off. There are also mappings with a latency equal to that of CoSA but with much greater energy which, however, allow for a much lower area occupation. These mappings can be chosen if the area is a priority. Figs. 4b and 4c show situations in which CoSA solution is Pareto efficient. In Fig. 4b the CoSA solution is the best solution in terms of latency, but MEDEA is able to find the best solution in terms of energy at the cost of a little increase in latency. There are also similar mappings in terms of latency that dramatically reduce area requirements at the cost of energy. In Fig. 4c CoSA solution is the best solution in terms of energy. For this workload, the proposed algorithm was able to find better solutions in terms of latency and area but worse in terms of energy. The designer should choose these solutions if area and latency are priorities. CoSA schedule is still a valid option if the area is not constrained.

B. Network Mapping Set Selection

When we schedule a layer l with MEDEA, we get a set $M_l = \{m_1, m_2, \ldots, m_P\}$ of Pareto optimal mappings to choose from. Thus, for a network with L layers, it needs to select a mapping set $C \in M_1 \times M_2 \times \cdots \times M_L$.

The number of possible mapping sets will be equal to the cardinality of the Cartesian product, namely P^L. For instance, VGG16 consists of 16 convolutional and fully connected layers. Assuming 100 Pareto mappings per layer, we have that the number of possible mapping sets will be 10^{32}.

TABLE II: End-to-end metrics percentage change with respect to CoSA solution for different mapping sets.

Net	Mapping Set	Area	Energy	Latency
VGG16	A	−87.39%	+4.85%	−5.52%
	B	−87.21%	−2.32%	+11.02%
Resnet50	A	−86.83%	+6.99%	−5.83%
	B	−77.00%	−2.54%	−0.25%
	C	−57.47%	−1.57%	−9.18%
	D*	−43.43%	−15.22%	−15.47%
DenseNet201	A	−81.49%	−9.36%	+4.04%
	B	−82.44%	−3.49%	−0.52%
	C	−82.04%	+25.52%	−16.59%
	D*	−50.47%	−2.60%	−19.55%
MobileNetV3	A	−83.41%	−14.91%	−24.10%
	B	−79.28%	−23.30%	−6.74%
	C*	−66.95%	−27.26%	−24.85%

* CoSA Solution injected

The designer would have to choose one from this gigantic amount of mapping sets that allows for the target inference metrics of the entire network to be met. To explore these combinations and compare them with the single mapping set obtained with CoSA, we first assume that layers having the same tensor dimensions will be executed according to the same mapping. For example, in VGG16, there are 12 unique levels in this sense. To further reduce the number of possible mapping sets, we preprocess the Pareto mappings obtained for each layer, considering only a portion of them. In particular, we fix an upper bound in latency and energy. For example, we consider only the individuals of the Pareto front belonging to the top-10 in terms of latency or energy for each layer. Eventually, we randomly search for mapping sets that maximize a score function defined as the linear combination of area, energy, and latency. We repeated this search with different score function coefficients to find mapping sets that exhibit unbalanced trade-offs in terms of metric improvements.

Table II reports the percentage change of end-to-end inference metrics compared to the respective CoSA solution

for different mapping set choices. The end-to-end energy and latency are calculated summing up the energy and latency of each layer in the network. End-to-end area is calculated considering the maximum chip area occupation among layers. In VGG16, we observe that it is possible to dramatically reduce area occupation, sacrificing either energy (Mapping Set A) or latency (Mapping Set B). In ResNet50, we can choose to save much chip area and improve latency sacrificing energy (Mapping Set A). We can also choose to save less area than (Mapping Set A) without jeopardizing other metrics. Lastly, we can choose to save even less area improving latency at the same time (Mapping Set C).

C. User Defined Individuals

As showed in Figs. 4b and 4c, there are some cases in which CoSA solution belongs to the Pareto front of the final MEDEA population. For this reason, we introduced the opportunity to inject user-defined individuals into MEDEA populations. This might be useful to explore mappings similar to the provided ones that exhibit unbalanced and favorable trade-offs but also favor the finding of overall better solutions.

We conducted some evaluations injecting CoSA solutions in MEDEA initial population. Doing so, overall better mapping sets are found. For example, in Table II we show that with mapping set D of ResNet50 it is possible to almost halve area occupancy while also decreasing both energy and latency by 15% with respect to the original CoSA solution. Injecting CoSA solution in Densenet201 mapping search unlocks the possibility to achieve better latency metric (D) without worsening energy consumption (C).

VI. Conclusion

This work presents MEDEA a multi-objective evolutionary approach for design space exploration of DNN hardware accelerators. MEDEA allows the co-exploration of the mapping space and the architectural space and provides a set of Pareto solutions featuring the best trade-off in terms of energy, latency, and area. Unlike the other techniques that provide a single solution, we show that, in practical cases, the availability of a set of Pareto solutions helps the designer in selecting the most appropriate system instance.

Future research should further improve the domain-specific genetic operators. An effort would also be needed to investigate better algorithms for choosing interesting network mapping sets for the designer. We also plan to extend MEDEA to work with Sparseloop [18] model for design-space and mapping exploration of sparse accelerators.

Acknowledgments

This work was supported by the Research Grants from: (a) the Italian Ministry of University and Research (MIUR) - PNR 2015–2020 within the projects (i) "4FRAILTY – Sensoristica intelligente, infrastrutture e modelli gestionali per la sicurezza di soggetti fragili" ref. ARS01_00345, and (ii) "MAIA – Monitoraggio Attivo dell'Infrastruttura", ref. ARS01_00353; (b) the University of Catania - Piaceri 2020-2022 - Linea 2, within the project MANGO.

References

[1] B. C. Edwards, "Moore's law: What comes next?" *Communications of the ACM*, vol. 64, no. 2, pp. 12–14, Feb. 2021.

[2] A. Parashar, P. Raina, Y. S. Shao, Y.-H. Chen, V. A. Ying, A. Mukkara, R. Venkatesan, B. Khailany, S. W. Keckler, and J. Emer, "Timeloop: A systematic approach to dnn accelerator evaluation," in *2019 IEEE international symposium on performance analysis of systems and software (ISPASS)*. IEEE, 2019, pp. 304–315.

[3] Q. Huang, A. Kalaiah, M. Kang, J. Demmel, G. Dinh, J. Wawrzynek, T. Norell, and Y. S. Shao, "Cosa: Scheduling by constrained optimization for spatial accelerators," in *2021 ACM/IEEE 48th Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 2021, pp. 554–566.

[4] K. Hegde, P.-A. Tsai, S. Huang, V. Chandra, A. Parashar, and C. W. Fletcher, "Mind mappings: enabling efficient algorithm-accelerator mapping space search," in *Proceedings of the 26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*, 2021, pp. 943–958.

[5] A. Symons, L. Mei, and M. Verhelst, "Loma: Fast auto-scheduling on dnn accelerators through loop-order-based memory allocation," in *2021 IEEE 3rd International Conference on Artificial Intelligence Circuits and Systems (AICAS)*. IEEE, 2021, pp. 1–4.

[6] S.-C. Kao and T. Krishna, "Gamma: Automating the hw mapping of dnn models on accelerators via genetic algorithm," in *2020 IEEE/ACM International Conference On Computer Aided Design (ICCAD)*. IEEE, 2020, pp. 1–9.

[7] Y. N. Wu, J. S. Emer, and V. Sze, "Accelergy: An architecture-level energy estimation methodology for accelerator designs," in *2019 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. IEEE, 2019, pp. 1–8.

[8] H. Kwon, P. Chatarasi, V. Sarkar, T. Krishna, M. Pellauer, and A. Parashar, "Maestro: A data-centric approach to understand reuse, performance, and hardware cost of dnn mappings," *IEEE micro*, vol. 40, no. 3, pp. 20–29, 2020.

[9] L. Mei, P. Houshmand, V. Jain, S. Giraldo, and M. Verhelst, "Zigzag: Enlarging joint architecture-mapping design space exploration for dnn accelerators," *IEEE Transactions on Computers*, 2021.

[10] V. Sze, Y.-H. Chen, T.-J. Yang, and J. S. Emer, "Efficient processing of deep neural networks: A tutorial and survey," *Proceedings of the IEEE*, vol. 105, no. 12, pp. 2295–2329, 2017.

[11] Y.-H. Chen, T. Krishna, J. S. Emer, and V. Sze, "Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks," *IEEE journal of solid-state circuits*, vol. 52, no. 1, pp. 127–138, 2016.

[12] S. M. Nabavinejad, M. Baharloo, K.-C. Chen, M. Palesi, T. Kogel, and M. Ebrahimi, "An overview of efficient interconnection networks for deep neural network accelerators," *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, vol. 10, no. 3, pp. 268–282, 2020.

[13] D. E. Goldberg and J. H. Holland, "Genetic algorithms and machine learning," *Mach. Learn.*, vol. 3, no. 2–3, p. 95–99, Oct. 1988.

[14] A. Konak, D. W. Coit, and A. E. Smith, "Multi-objective optimization using genetic algorithms: A tutorial," *Reliability engineering & system safety*, vol. 91, no. 9, pp. 992–1007, 2006.

[15] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan, "A fast and elitist multiobjective genetic algorithm: Nsga-ii," *IEEE transactions on evolutionary computation*, vol. 6, no. 2, pp. 182–197, 2002.

[16] E. Russo, M. Palesi, S. Monteleone, D. Patti, G. Ascia, and V. Catania, "Lambda: An open framework for deep neural network accelerators simulation," in *IEEE International Conference on Pervasive Computing and Communications Workshops and other Affiliated Events (PerCom Workshops)*, 2021, pp. 161–166.

[17] Y. S. Shao, J. Clemons, R. Venkatesan, B. Zimmer, M. Fojtik, N. Jiang, B. Keller, A. Klinefelter, N. Pinckney, P. Raina *et al.*, "Simba: Scaling deep-learning inference with multi-chip-module-based architecture," in *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture*, 2019, pp. 14–27.

[18] Y. N. Wu, P.-A. Tsai, A. Parashar, V. Sze, and J. S. Emer, "Sparseloop: An analytical, energy-focused design space exploration methodology for sparse tensor accelerators," in *2021 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*. IEEE, 2021, pp. 232–234.