SENNA: Unified Hardware/Software Space Exploration for Parametrizable Neural Network Accelerators

JUNGYOON KWON, Dept. of Computer Science and Engineering, Seoul National University, Seoul, South Korea
HYEMI MIN, Dept. of Computer Science and Engineering, Seoul National University, Seoul, South Korea
BERNHARD EGGER, Dept. of Computer Science and Engineering, Seoul National University, Seoul, South Korea

Parametrizable neural network accelerators enable the deployment of targeted hardware for specialized environments. Finding the best architecture configuration for a given specification, however, is challenging. A large number of hardware configurations have to be considered, and for each hardware instance, an efficient software execution plan needs to be found, leading to a vast search space. Prior work has tackled this problem by dividing the search into subproblems for individual layers of a network. There is no guarantee, however, that the overall best hardware configuration that delivers the desired end-to-end performance across the entire network is among the best individual layer configurations.

This work presents SENNA, a unified hardware/software space exploration framework for parametrizable neural network accelerators. To guide the exploration toward the overall best configuration, SENNA employs a multi-objective genetic algorithm with a novel design space representation that encodes the configuration of hardware and software parameters in a single chromosome. Using the Parallel Island Model (PIM), each layer is represented by one or more individual islands each containing a separate population to simultaneously search for the best configuration across the entire network. A tailored gene migration technique enables the exchange of genes between the populations of different islands.

SENNA is evaluated with three parametrizable architectures and four neural networks. The evaluation result demonstrates that SENNA achieves upto 1.92x EDP improvement compared to the State-of-the-Art. With equivalent evaluation budgets, SENNA shows 2.5x–9.3x speedup compared to an Oracle scheme and the State-of-the-Art.

CCS Concepts: • **Hardware** → **Electronic design automation**; **Physical design (EDA)**; **Placement**;

Additional Key Words and Phrases: HW/SW co-design, reconfigurable accelerators, design space exploration, multi-objective genetic algorithm, parallel island model

This work was funded, in parts, by the Korean National Research Foundation through grants 21A20151113068 (BK21 Plus for Pioneers in Innovative Computing—Dept. of Computer Science and Engineering, SNU), 10077609 (MOTIE/KEIT), and RS-2023-00302083 (MSIT). ICT at Seoul National University provided research facilities for this study.

Authors' Contact Information: Jungyoon Kwon, Dept. of Computer Science and Engineering, Seoul National University, Seoul, South Korea; e-mail: jungyoon@csap.snu.ac.kr; Hyemi Min, Dept. of Computer Science and Engineering, Seoul National University, Seoul, South Korea; e-mail: hyemi@csap.snu.ac.kr; Bernhard Egger (corresponding author), Dept. of Computer Science and Engineering, Seoul National University, Seoul, South Korea; e-mail: bernhard@csap.snu.ac.kr.

ACM 1539-9087/2025/01-ART30
https://doi.org/10.1145/3705731

ACM Reference Format:
Jungyoon Kwon, Hyemi Min, and Bernhard Egger. 2025. SENNA: Unified Hardware/Software Space Exploration for Parametrizable Neural Network Accelerators. *ACM Trans. Embedd. Comput. Syst.* 24, 2, Article 30 (January 2025), 26 pages. https://doi.org/10.1145/3705731

1 Introduction

Over the past decade, we have witnessed unprecedented advances in deep learning. In multiple areas, artificial intelligence powered by deep learning now outperforms traditional algorithms and even human experts, such as, for example, in image recognition, natural language processing, and game play [27, 28, 45]. The **deep neural networks (DNNs)** that made this achievement possible have become increasingly versatile and operate seamlessly across a wide range of devices from large data centers to resource-constrained IoT devices.

In embedded systems, the execution of DNN workloads faces significant challenges due to their high computational complexity and energy consumption [6]. To address this problem, researchers have explored software optimizations such as network pruning [35], bit-width reduction [12], and network reorganization [48]. On the hardware side, tailored accelerators have been proposed to improve performance and energy efficiency in embedded systems [1, 8, 11]. Recently, reconfigurable neural accelerators that allow the optimization of hardware parameters towards specific DNN workloads have shown particular promise [17, 38, 40].

Reconfigurable accelerators are parametrizable in terms of the compute and memory resources, I/O bandwidth, and **network-on-chip (NoC)** connectivity at design time. Determining the hardware parameters of a reconfigurable accelerator to obtain a configuration that meets specific constraints is, however, a complex task. The performance of an accelerator cannot be inferred directly from its hardware parameters and requires sophisticated algorithms to find the best execution schedule for a given configuration [33, 51]. For example, the workload of a convolution operator typically has to be tiled because the layer data does not fit into the on-chip memory. The dimensions along which a workload is tiled and the execution order of the tiled operations strongly affect the execution time and amount of data transferred between on- and off-chip memories [11], hence, finding the optimal tiling parameters that minimize energy consumption and maximize performance across all layers of a network is a significant challenge in itself [34].

Previous studies have focused on co-optimizing the hardware (architecture configuration) and software (tiling and data flow order) design space. To tame the vast search space, each layer is optimized separately, resulting in repeated searches for each individual layer [26, 41, 42, 54, 55, 58]. For a network with N layers, the design optimization search is repeated N times and potentially yields N discrete hardware configurations. These discrete configurations are not guaranteed to include the optimal configuration for the entire network. As a result, identifying the unified hardware configuration that achieves the overall best performance for an entire network remains an open problem.

To address these challenges, this work presents SENNA, a unified hardware/software **S**pace **E**xploration framework for parallel **N**eural **N**etwork **A**ccelerators. SENNA employs the **Parallel Island Model (PIM)** to simultaneously perform a **design space exploration (DSE)** for the best configurations of each layer as well as the overall network. In PIM, each layer is represented by one or more islands. The evolution of the population is implemented by a customized multi-objective genetic algorithm that combines the encoding of the hardware and software configuration of a layer into a single chromosome. By exchanging the best-performing chromosomes between islands, the algorithm converges towards globally optimal configurations. Since the parameters of a layer differ from island to island, in general, the chromosomes of one island constitute an invalid configuration on a different island. To ensure that only valid hardware configurations are exchanged, SENNA employs a tailored migration algorithm that shares only the hardware portion of a chromosome between islands. Evaluated with three parametrizable architectures and four networks, SENNA is able to find a better hardware configuration than state-of-the-art techniques in significantly less time.

In summary, the main contributions of this article are as follows:

— We present a unified encoding that embeds the hardware configuration and the software mapping of a layer in a single chromosome.
— We employ a **multi-objective evolutionary algorithm (MOEA)** on a PIM with tailored migration operations to simultaneously search for locally and globally optimal configurations.
— A comparison with an Oracle scheme and the state-of-the-art shows that SENNA is able to find significantly better configurations for networks in considerably less time.

The remainder of this article is organized as follows. Section 2 presents the background of this work and reviews related approaches. The design and implementation of SENNA are discussed in Section 3. Section 4 presents an evaluation of SENNA compared to baseline search methods. Section 5, finally, concludes this work.

2 Background

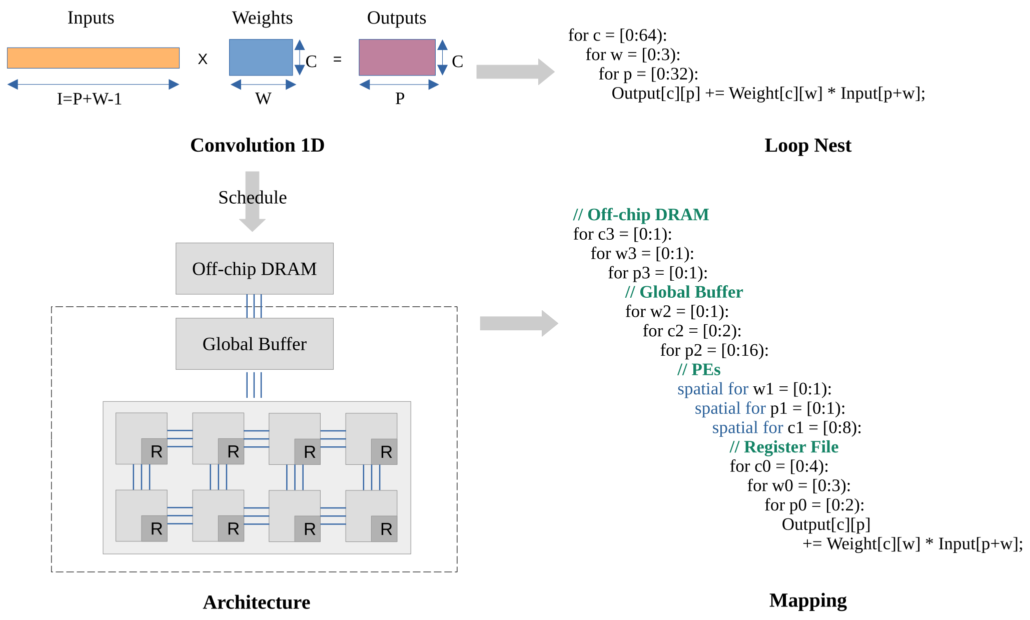
2.1 Execution of DNN Workloads on Accelerators

Neural networks are growing deeper and feature more and more layers. Popular large networks perform billions of operations on gigabytes of data—data that must be loaded into the accelerator to run network inference at a decent speed [6, 7, 31, 49]. This poses a particular challenge in the embedded systems world and led to the desire to design tailored hardware accelerators that can execute specific neural networks with given chip area, power, or latency constraints.

Both academia and industry have presented neural network accelerators that achieve high energy efficiency and throughput [8, 11, 20, 44]. Eyeriss [11] explores different orders of dataflows, including input-, weight-, output-, and row-stationary dataflows, to achieve efficient execution. The DianNao accelerators [8, 18, 32] minimize memory transfer between on- and off-chip memory. NVDLA [60] is an architecture optimized for weight-stationary dataflow, and Tetris [20] uses bypass ordering, which is similar to data stationary optimization. Simba [44] extends an accelerator with multiple chiplets considering non-uniform latency.

The high degree of configurability and control leads to a large number of possible execution orders (mappings) of a DNN workload to a given accelerator design. The popular **convolutional neural networks (CNNs)** are mainly composed of convolutional and fully-connected layers that dominate the overall runtime [10]. The basic computations of these layers are matrix multiplications that can be described with nested loops [39, 54]. For example, the computation of a convolutional layer can be expressed as a seven-fold nested loop that takes a three-dimensional input and four-dimensional filters as inputs to compute a three-dimensional output feature map. A *mapping* refers to an operation schedule that executes a loop nest on an accelerator. Since the organization of the loops dictates the sequence of the data-access patterns, a schedule must consider the storage hierarchy of the accelerator and the capabilities of its computational resources to explore possible data reuse opportunities. Loop optimization techniques such as loop unrolling, loop tiling, and loop interchange have been used to find the data partitioning and execution order that yields an optimal execution schedule [14].

Figure 1 illustrates the mapping of a 1D convolutional layer to a spatial architecture with a 3-level storage hierarchy composed of external DRAM, an on-chip buffer, and eight **processing elements (PEs)** with private register files. The 1D convolutional layer is represented as a nested



```
for c = [0:64):
  for w = [0:3):
    for p = [0:32):
      Output[c][p] += Weight[c][w] * Input[p+w];
```

```
// Off-chip DRAM
for c3 = [0:1):
  for w3 = [0:1):
    for p3 = [0:1):
      // Global Buffer
      for w2 = [0:1):
        for c2 = [0:2):
          for p2 = [0:16):
            // PEs
            spatial for w1 = [0:1):
              spatial for p1 = [0:1):
                spatial for c1 = [0:8):
                  // Register File
                  for c0 = [0:4):
                    for w0 = [0:3):
                      for p0 = [0:2):
                        Output[c][p]
                          += Weight[c][w] * Input[p+w];
```

Fig. 1. Mapping example of a 1D convolutional layer with C = 64, W = 3, and P = 32.

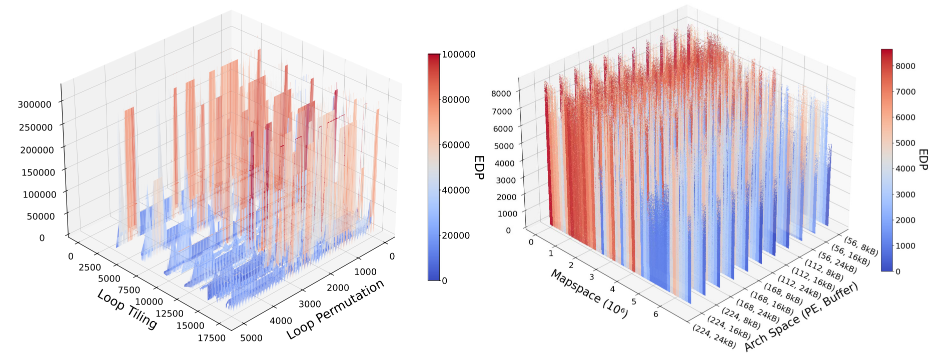
loop iterating over the width of the weights (W) or the output (P), and the height of the kernel and feature map (C). The simple 3-fold nested loop of the 1D convolution is explicitly tiled into a 12-fold nested loop to exploit spatiotemporal data reuse opportunities presented by the storage hierarchy and the parallelism in the hardware. The mapping of Figure 1 shows only one of many possible execution schedules. Other schedules may lead to shorter execution times or minimize the accesses to the on-chip memory or external DRAM. Even for a single hardware configuration, these spatiotemporal data reuse opportunities form a large mapspace of potential execution schedules.

2.2 Search Methods

Finding the single hardware configuration that leads to optimal performance across all layers requires an exploration of many different hardware configurations for each layer, for each of which a large number of possible execution schedules exist. In practice, the search space is too large for a brute-force evaluation of all possible combinations.

Existing accelerator performance evaluation tools can be classified into three types according to how they address the scheduling problem. Heuristic search algorithms prune the mapspace based on prior knowledge and use an analytic model for the evaluation. Exhaustive search [53], random search [29, 39], and beam search [2] are among the proposed search algorithms. Because these approaches require a large amount of computational resources, they are able to only explore a limited part of the entire mapspace and therefore often end up with a suboptimal solution. Black-box optimization techniques such as Simulated Annealing [9], Bayesian optimization [36, 52], and Genetic algorithms [25] were presented to overcome the complexity and infeasibility of exploring the entire search space. Other feedback-based techniques such as Reinforcement Learning [59] or constrained-based methods [22, 23] were also suggested as alternatives.

To automate the accelerator design process for a specific CNN model, prior works encompass DSE engines and accelerator modeling tools [52, 55, 58]. Bayesian optimization [52] and Particle Swarm Optimization [58] are two methods proposed to explore the vast search space more



(a) Mapspace (b) Joint Design Space

Fig. 2. Cost surface plot of Mapspace and Joint Design Space on the first layer of MobileNet-V2. Darker blue indicates lower EDP. Joint Design Space illustrates the cost surface for running the given layer on a number of Eyeriss architecture designs. Interdependent subspaces form a broaden search space and show a non-convex and non-smooth cost surface.

efficiently. Beyond accelerator-mapping co-design, similar approaches are also found in the area of **Neural Architecture Search (NAS)** that conducts HW-aware Neural Network optimization [19, 30].

In this work, we employ a multi-objective genetic algorithm to guide the search. Genetic algorithms are a popular optimization technique in which a population of candidate solutions evolves over several generations (epochs) toward a better solution. The parameters of each member of the population are encoded in a chromosome, and a fitness function evaluates the quality of a member. Some candidates are selected and modified during evolution by applying crossover and mutation operations on the chromosomes. The next generation is typically composed of well-performing candidates, mutated candidates, and newly injected candidates. A genetic algorithm evolves over generations until it converges to a solution or reaches a limit on the maximum number of generations.

2.3 Related Work

The cartesian product of the valid hardware configurations and possible software execution plans form a vast non-convex search space with many local minima and invalid points [22]. To provide a clear view of the variety of design choices available, Figure 2 shows the cost surface of mapspace and joint design space for the Eyeriss architecture running the first layer of MobileNet-V2. Two subspaces are included on mapspace, defined from the loop tiling choices and loop permutation choices. Together, they deliver different costs of mapping and form a non-convex and non-smooth search space. Joint Design Space shows that the search space quickly increases adding a number of hardware. Practical designs are more complicated and open a new challenge to explore this broaden search space.

Motivated by the search methods described in the previous section, early works attempt to co-optimize the accelerator configuration with software mappings to obtain a solution. Table 1 lists and compares the most relevant related works. HASCO [54] employs multi-objective Bayesian optimization for the hardware DSE and a Q-learning algorithm to explore the software mapspace. Confuciux [24] is based on reinforcement learning and an evolutionary algorithm for higher accuracy. The development cost and turnaround time of both approaches are prohibitively high because the search of the two design spaces is handled separately in a nested optimization loop (Figure 3),

Table 1. State-of-the-Art HW-SW Exploration Frameworks

Framework	*Joint HW-SW Optimization*	*Multiple Objectives*	*Network-wide Exploration*	*Network-wide HW Search*
HASCO [54]		✓		
Confuciux [24]		✓	✓	
DiGamma [26]	✓			
MEDEA, EPOCA [41, 42]	✓	✓	✓	
SENNA (this work)	✓	✓	✓	✓

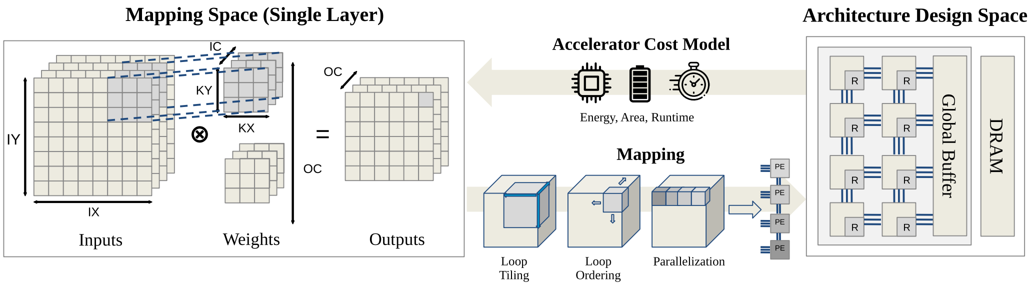


Fig. 3. Joint hardware/software DSE on limited mapspace.

i.e., the software mapping is evaluated separately for each hardware configuration. Minor changes in the accelerator design invalidate the prior knowledge of mappings and require a new full DSE.

Integrating the two searches into a single DSE process enhances the efficiency of finding the best architecture design for a given layer. Digamma [26] combines the exploration of the hardware search and software mapping in an evolution algorithm with a sequential encoding of genes. MEDEA [41] and EPOCA [42] first find feasible software mappings for each layer without considering the hardware. Then, they pair each mapping with the amount of resources exercised on an architecture instance with abundant resources and employ a multi-objective genetic algorithm to find the best configuration for each layer. The global hardware configuration A_N which delivers the overall performance of $z*$ is obtained by selecting the maximum of each configurable hardware resource a_n across the local (layer) configurations (a_n,m_n). That is,

$$z* = \Sigma f_n(A_N, m_n), n = 1, 2, \ldots, N$$

$$A_N = max(a_n), n = 1, 2, \ldots, N,$$

where the cost function $f_n(a_n, m_n)$ denotes the cost of executing software mapping m_n on hardware configuration a_n for layer n. MEDEA [41] suggests heuristics that choose the mappings based on multiple score functions. EPOCA [42] replaces the heuristics with an evolutionary algorithm that selects the best hardware/software configuration for each layer. The global hardware configuration A_N is derived from the local optima for each layer, and the metrics, such as energy or latency, are recomputed with the hardware configuration A_N and aggregated to obtain the overall performance.

While these techniques achieve satisfactory results for specific accelerators, only a small part of the vast hardware and software design space is explored. Both DiGamma [26] and Confuciux [24] make use of Maestro [29], a software mapper that is limited to weight-stationary data flows with private L1 and global L2 scratchpad memories [56], to generate software schedules. The limited configurability of Maestro hinders the application of the methods towards other reconfigurable hardware architectures. More importantly, no existing technique performs a global search on the entire network but rather generates the global solution from the set of locally optimized configurations with limited mapspaces. The Pareto-optimal solutions for each layer form another combinatorial optimization problem to find the best hardware and software configuration. To support the execution of every locally optimized mappings, the globally optimized hardware configuration is composed of each maximum HW component size from the set of locally optimized configurations. However, the proposed design tends to be area-intensive and easily leads to tedious explorations and sub-optimal solutions. To address the challenges above, this work proposes a novel design space representation that embraces both design spaces, so that the search method can conduct an efficient and flexible choice on hardware configurations. It facilitates the exploration of the architecture design that requires a minimum amount of resources with minimal degradation in performance.

2.4 Multi-Objective Evolutionary Algorithms

In multi-objective optimization problems, we consider more than one objective function to be maximized or minimized in the presence of certain constraints. Mathematically, the problem addressed in this article is to find the ideal solution z* that is the minimization of multiple cost functions $f_m(x)$ with lower and upper bounds on the problem variables:

$$z* = argmin f_m(x), m = 1, 2, \ldots, M$$
$$x_i^{(L)} \leq x_i \leq x_i^{(U)}.$$

While the superiority of a solution in a single-objective is found by comparing cost function values, the superiority in a multi-objective is determined based on Pareto dominance relations. With conflicting objectives, typically, no point maximizes all the objectives simultaneously. The Pareto front or Pareto-optimal set represents the collection of solutions where no other solution can improve on at least one objective without sacrificing performance on another objective. A multi-objective optimization problem thus aims at approximating the Pareto-optimal set.

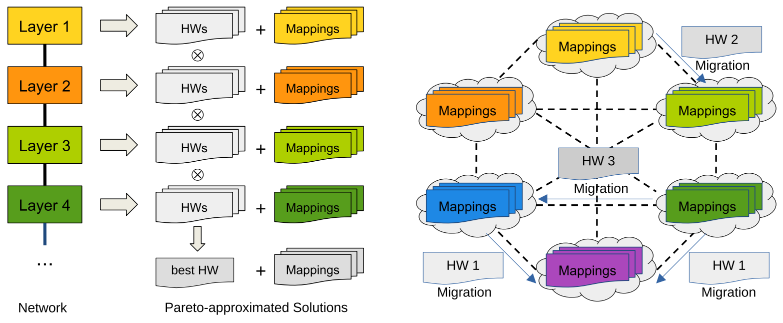
In addition to the Pareto set, the ideal and the nadir points are estimated during the optimization. These points represent the best, respectively, the worst objective value over the Pareto-optimal set with respect to a given objective. The ideal and the nadir point thus define the possible value ranges of the objective functions and are used to guide the search, normalization, and visualization [3, 15].

MOEAs [16, 61] have been applied successfully to a number of problems with multiple objectives because of their ability to explore a large global search space and obtain an evenly and well-distributed Pareto-front. NSGA-II [16] is one of the most popular MOEAs that employ a fast nondominated sorting algorithm when ranking solutions. MOEAD [57] uses decomposition to divide the problem into single objective subproblems and optimizes the subproblems concurrently.

2.5 Parallel Island Mode

A combinatorial optimization problem is an NP-hard problem that finds the best solution in a finite or possibly countably infinite set [5]. Since the search space grows super-exponentially, an exhaustive search is infeasible. Metaheuristics are widely acknowledged tools to address this complex problem with a finite set of solutions but are still limited in achieving a reasonable computing time.

Parallel metaheuristics aim at addressing both of these problems. The PIM, also known as an *archipelago*, was initially proposed for genetic algorithms in order to improve the quality of solutions and reduce the runtime of the algorithms [13]. In a PIM, the global population is divided into "islands" containing distinct subpopulations that evolve in parallel. A topology defines the connections between the islands and allows the exchange of individuals through migration. Evolution is augmented by migration to increase the likelihood of finding globally optimal solutions since the



(a) Previous approaches (b) SENNA

Fig. 4. Previous approaches divide the search into layer-wise subproblems and output a hardware obtained through a combination of the per-layer best configurations. SENNA, on the other hand, searches for the global best hardware by employing a PIM.

exchange of locally optimized solutions from different islands introduces more diversity [47]. The topology of an archipelago can be static or dynamic during the evolution. Different metaheuristics, such as genetic algorithms, particle swarms, simulated annealing, or colony optimization, can be used as the evolutionary engine.

3 Design and Implementation

This section discusses the formulation of the design space and the entailed optimizations to obtain a feasible search space. We employ a novel representation of the design space that is suitable to guide the DSE toward the global best configuration of a given DNN model. To the best of our knowledge, this work is the first to formulate and explore global DSE for parametrizable neural network accelerators.

3.1 Challenges and Motivation of Using the Parallel Island Model

Finding the best architecture design for a given DNN model is particularly challenging since evaluating the entire model is expensive and time-consuming. Prior work has tackled this problem by dividing it into subproblems for the layers of a network [24, 54]. Figure 4(a) shows this approach. Solutions comprising pairs of hardware configurations and software mappings are generated for every layer. Finding the best single overall hardware configuration and per-layer software mappings from the Pareto-approximated solutions of the individual layers forms another exploration of a non-trivial search space. Choosing the global hardware configuration from a set of per-layer Pareto sets can be formulated as a combinatorial optimization problem; however, the best hardware configuration of one layer may be suboptimal for another layer. We also need to consider that some hardware configurations in the Pareto set of one layer may not exist in other layers, requiring additional explorations of the software mapspace.

To this end, a systematic approach is required that integrates layer-wise searches into a global DSE. We propose to leverage the idea of migration from the PIM to exchange the best hardware configurations between different layers. The benefit of this approach is that the locally best candidates are shared with other islands, leading to a convergence on the globally best configuration.

Traditional implementations of PIMs assume compatibility of chromosomes between island and simply migrate entire chromosomes. In our scenario, however, chromosomes of different islands may not be compatible with each other because a software mapping is tied to a given

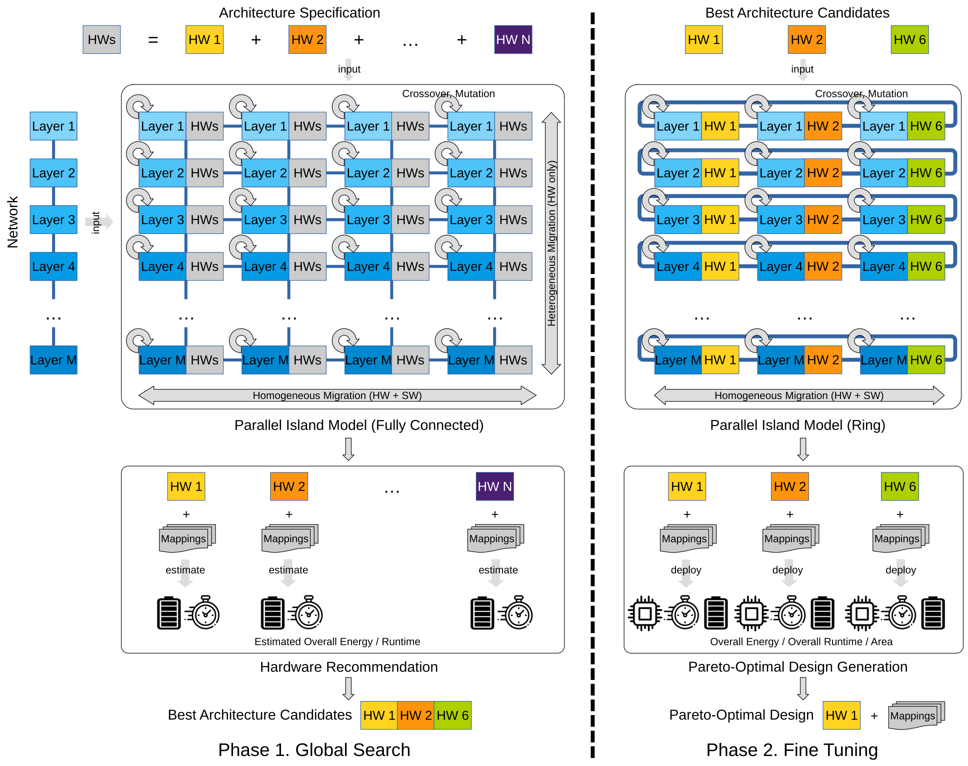

Architecture Specification
HWs = HW 1 + HW 2 + … + HW N
input
Network
Layer 1
Layer 2
Layer 3
Layer 4
Layer M
Crossover, Mutation
Heterogeneous Migration (HW only)
Homogeneous Migration (HW + SW)
Parallel Island Model (Fully Connected)
Mappings
estimate
Estimated Overall Energy / Runtime
Hardware Recommendation
Best Architecture Candidates HW 1 HW 2 HW 6
Phase 1. Global Search
Best Architecture Candidates
Parallel Island Model (Ring)
deploy
Overall Energy / Overall Runtime / Area
Pareto-Optimal Design Generation
Pareto-Optimal Design HW 1 + Mappings
Phase 2. Fine Tuning

Fig. 5. Overview of the SENNA methodology.

hardware configuration and the layer parameters. While the hardware configuration is a feature of the parametrizable hardware, a specific configuration is always valid on any island, however, the layer parameters of the migrated-to islands may not be valid. For instance, the operators of different layers vary in the dimensions of the input, weight, and output parameters and thus require a re-exploration of the specific parameter space. The problem can be considered a heterogeneous optimization problem with incompatible encodings caused by different layer parameters. Such incompatible encodings not only increase the complexity of the search but also lead to a corruption of the search space. Therefore, we do not migrate the entire chromosome. Instead, only the hardware configuration part of the chromosome is migrated (Figure 6(b)), and a compatible software mapping is computed on the destination island. If no mapping can be found because the hardware configuration is incompatible with the layer parameters, the migrated chromosome is discarded. This approach ensures that only compatible hardware configurations are exchanged while preserving diversity and efficient mappings on each island.

SENNA uses a two-phase approach to find a global solution (Figure 5). In the first phase, called the *global search* phase, SENNA employs a multi-objective genetic algorithm on a PIM to perform the DSE. The population is encoded with architecture design and mapspace choices and divided into subpopulations on the islands of PIM. The proposed migration operators and the PIM topology enable the algorithm to converge towards the globally optimal architecture design and increase the likelihood of escaping local optima. Moreover, we develop a method to estimate the overall performance metrics based on Pareto-approximated solutions to select a suitable architecture configuration as detailed in Section 3.4. In the second phase, called the *fine tuning* phase, the chosen architecture design from the global search phase is further optimized to improve the quality of the mappings. At the end of the second phase, the overall performance statistics are used to determine the best architecture design and the corresponding per-layer software mappings.

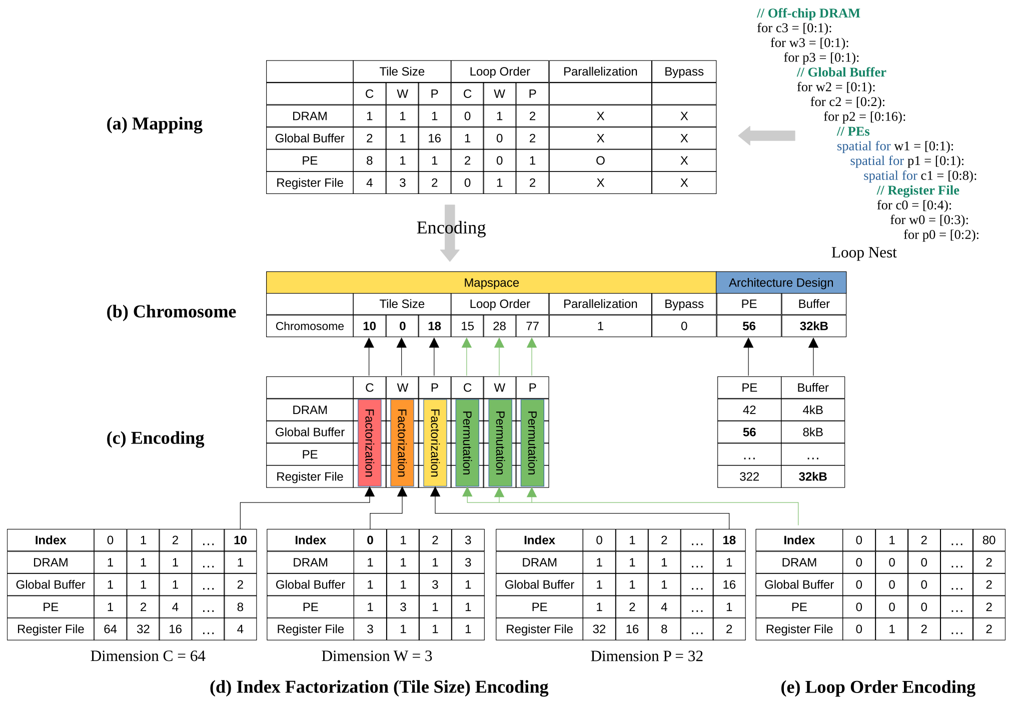
3.2 Encoding Design Space Parameters

Here, we first describe how SENNA represents the different design parameters. To support the layer-aware migration described in the previous section, dedicated encoding and specialized operators are necessary. While prior work typically encodes the bounds of the deeply-nested loops with respect to the memory hierarchy; however, this leads to long encodings that are proportional to the depth of the memory hierarchy. SENNA, on the other hand, employs a compact encoding with a fixed length that well fits emerging accelerators and networks. SENNA's software mapspace that encodes the tiling size, the order of the data flow, and loop unrolling parameters are represented by the following four subspaces:

— The **tile size** represents the factorization of the loop bounds across the memory hierarchy.
— The **loop order** represents a permutation of the loop nests on each level of the memory hierarchy.
— The **parallelization** parameter indicates whether the innermost loop can be distributed (unrolled) to parallel hardware components.
— The **bypass** parameter indicates whether the hardware has dedicated on-chip memories for the different types of data (`bypass=1`) or whether input, weight, and output data are allocated to a shared buffer (`bypass=0`).

The `parallelization` and `bypass` subspaces can each be represented by a single gene. The `tile size` and `loop order` subspaces encode large parameter spaces that reach up to $O(10^{62})$ for the Eyeriss architecture with the VGG16 network. For a 2-dimensional convolution, the two subspaces each comprise seven elements to represent the tile size and loop level of each of the seven nested loops and are encoded as integer values. For a 1-dimensional convolution, as shown in Figure 1, each space requires three genes. The software mapspace thus requires 8 (=3+3+1+1) and 16 (=7+7+1+1) genes, respectively, to express 1- and 2-dimensional convolutions. After we split the `tile size` and `loop order` subspaces according to the number of loop dimensions, the possible factorizations of each loop bound in the corresponding memory hierarchy level are encoded as integer values. We apply a lexicographical order from the outermost to the innermost level of the memory hierarchy.

We refer to the mapping example from Figure 1 and describe the joint encoding scheme of the design spaces in Figure 6. Figure 6(a) applies the encoding to represent the information of a software mapping. A 1-dimensional convolutional layer has three loop variables C, W, and P with bounds 64, 3, and 32. Loop variable C with a bound of 64 is factorized and allocated to the four levels in the memory hierarchy with factors (1,2,8,4). Since the loops in DRAM are mapped $C \rightarrow W \rightarrow P$ (top right of Figure 6(a)), the encoding is 0 (representing C), 1 (W), and 2 (P). Each possible distribution of the loop bound is encoded as an integer value as depicted in Figure 6(c). For each dimension, the enumeration of all possible factorizations is found and sorted in ascending order from DRAM to the Register File. This is done for each loop variable C, W, and P with different encoding representations. Figures 6(d) and (e) illustrate the encoding of tilings and loop orders. After a loop order has been determined for each level in the memory hierarchy, it is repartitioned by the dimension and encoded in ascending order of storage level. The genes `parallelization` and `bypass` are encoded with 1 and 0, respectively, as the given architecture maps the data spatially across the PEs and accommodates all types of data on each level of the memory hierarchy.



	Tile Size			Loop Order			Parallelization	Bypass
	C	W	P	C	W	P		
DRAM	1	1	1	0	1	2	X	X
Global Buffer	2	1	16	1	0	2	X	X
PE	8	1	1	2	0	1	O	X
Register File	4	3	2	0	1	2	X	X

```
// Off-chip DRAM
for c3 = [0:1):
  for w3 = [0:1):
    for p3 = [0:1):
      // Global Buffer
      for w2 = [0:1):
        for c2 = [0:2):
          for p2 = [0:16):
            // PEs
            spatial for w1 = [0:1):
              spatial for p1 = [0:1):
                spatial for c1 = [0:8):
                  // Register File
                  for c0 = [0:4):
                    for w0 = [0:3):
                      for p0 = [0:2):
```

Mapspace									Architecture Design	
	Tile Size			Loop Order			Parallelization	Bypass	PE	Buffer
Chromosome	**10**	**0**	**18**	15	28	77	1	0	**56**	**32kB**

	C	W	P	C	W	P
DRAM						
Global Buffer						
PE						
Register File						

PE	Buffer
42	4kB
56	8kB
…	…
322	**32kB**

Index	0	1	2	…	**10**
DRAM	1	1	1	…	1
Global Buffer	1	1	1	…	2
PE	1	2	4	…	8
Register File	64	32	16	…	4

Index	**0**	1	2	3
DRAM	1	1	1	3
Global Buffer	1	1	3	1
PE	1	3	1	1
Register File	3	1	1	1

Index	0	1	2	…	**18**
DRAM	1	1	1	…	1
Global Buffer	1	1	1	…	16
PE	1	2	4	…	1
Register File	32	16	8	…	2

Index	0	1	2	…	80
DRAM	0	0	0	…	2
Global Buffer	0	0	0	…	2
PE	0	0	0	…	2
Register File	0	1	2	…	2

Fig. 6. Gene encoding example of the 1-dimensional convolution layer from Figure 1.

The encoding of the hardware parameters follows the encoding of the software mapspace and represents the number or size of each hardware component. Figure 6(b) shows the parameter space for an architecture with a parametrizable number of PEs and a configurable size of a global buffer memory. The size of the input, weight, and accumulation buffer and the memory bandwidth can be additionally configured.

The presented approach combines the hardware configuration and the software mapspace into a single chromosome to obtain a joint encoding. It is worth noting that the encoding does not incur any constraints on the dependency between hardware configurations and software mappings. For example, the validity of a software mapping depends on the size of each hardware component of a given configuration. Invalid encodings are dealt within the evaluation phase.

3.3 Network Formulation

Neural networks can be broken down into discrete layers based on the employed operators. The computation of each operator, such as 2D convolutions, depthwise convolutions, or fully-connected layer, can basically be expressed with a matrix multiplication with different dimensions. It may be desirable to represent the mapspace of the different operators with unequal encoding lengths. Moreover, the different dimensions of individual layers require a separate exploration of the mapspaces, even if the computational operator is the same. Thus, we consider the mapping of layers as a set of heterogeneous problems that require different sizes and types of encodings to fully explore the entire mapspace.

On the other hand, a unified construction of the software mapspace and the architecture design space is required to guide the search toward the globally optimal solution. To deal with the varying properties of different layers yet enable a global search, we employ and extend the PIM to fulfill the following characteristics:

— Islands shall allow distinct encodings to express heterogeneous problems.
— The PIM assembles the heterogeneous islands into a unified mapspace.
— Migration between heterogeneous islands shall not violate the predefined encodings.

The following properties make PIMs suitable for our search problem:

— The gene encoding of the software mapping depends on the operator of a given layer while the hardware configuration is shared across all islands.
— The topology connects the individual layers' mapspaces and introduces population diversity through migration.

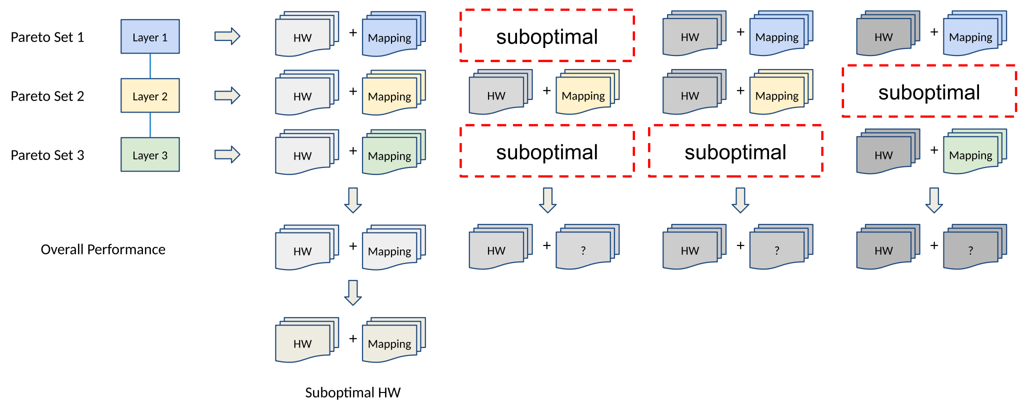
The first property implies that the software mapping is local information that is only valid on a specific layer while the hardware configuration is global information. In other words, the incompatibility between different islands can be solved by limiting migration to exchange hardware configurations only. This allows each island to preserve its diversity and maintain valid and efficient mappings, while the hardware configurations can be shared through the topology of the PIM. This fits a network comprised of multiple operators with variable lengths of encodings into a single PIM.

The second property is crucial in finding the best hardware configuration across the network. During the evolution of the PIM, the best-performing hardware configurations are shared through the PIM topology and distributed to multiple layers. This ensures that promising candidates in the hardware design space have a higher chance of being chosen as a migrant and guides the search towards the global optima.

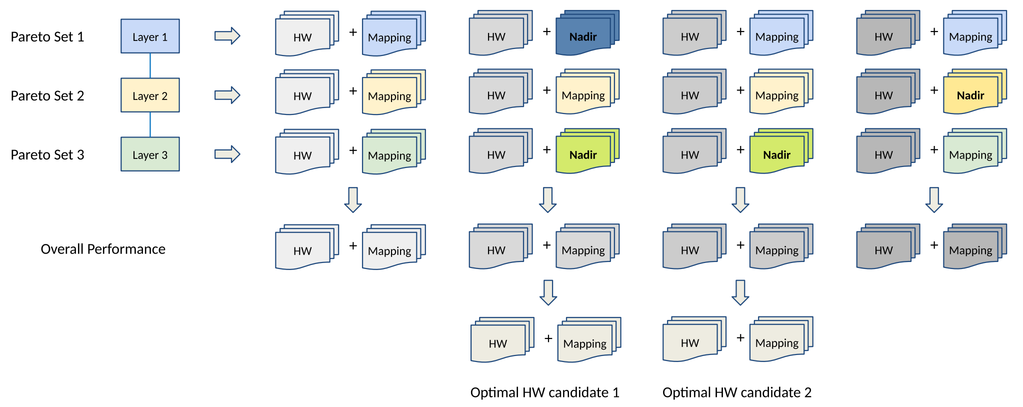
3.4 Hardware Recommendation

A major challenge of the architecture DSE is the sparsity of the Pareto-approximated solutions due to the large dimensionality of the design space and limited budget of evaluations. This is shown in more detail in Figure 7(a). On each island of the PIM, Pareto-approximated solutions for a given layer are found that encode the hardware and software mapping as described earlier. The Pareto-approximated set of potential solutions does include configurations that are suboptimal or can miss configurations on the Pareto set because of the limited exploration of the parameter space. The more heterogeneous islands (i.e., layers) a network is composed of, the lower the likelihood that a specific hardware configuration is present in the approximated Pareto fronts of all individual islands.

Previous approaches heuristically derive the final architecture by choosing the maximum of each individual hardware parameter from the set of layers and obtaining the overall performance of the network by computing new software mappings for this all-encompassing architecture [41, 42]. Such an approach, however, tends to lead to area-intensive and energy-inefficient configurations. In the presented work, we instead conduct a global hardware configuration exploration by estimating the overall performance of a given architecture design based on the Pareto-approximated solutions. Since the multi-objective algorithm guides the search, the range of the objectives has already been determined by the ideal and the nadir point. This enables us to estimate the value of missing objectives to a reasonable value (Figure 7(b)). In other words, SENNA estimates the performance metrics of unknown architectures to be no worse than the so-far observed worst performance, i.e., the nadir point. These estimates are automatically updated if a given hardware configuration is later discovered to be in the Pareto-approximated set. Estimating performance metrics from nadir points can be imprecise if the Pareto set is sparsely populated. Such a situation can occur when the search is not thoroughly conducted on the initial generation or the design space is filled with a large portion of infeasible points. In such a case, the missing objectives are compensated by the worst observed values from another layer's Pareto-approximated set.



(a) Pareto solutions with missing design points



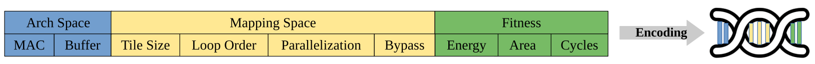
(b) Compensation of suboptimal design points with nadir points

Fig. 7. Estimation of overall performance through nadir points.

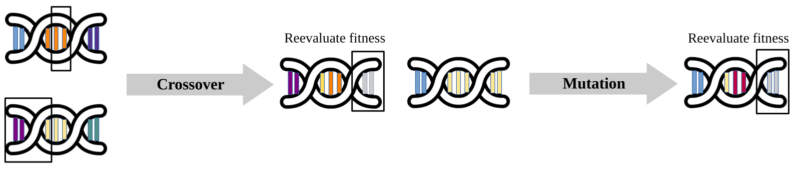
The estimation of the overall performance not only exhibits a convergence towards the Pareto-optimal solutions on individual layers but also approaches the global best hardware configuration for a given network. SENNA monitors the convergence of the PIM according to the estimated overall performance and chooses the candidates of potentially good hardware configurations. The fine-tuning phase is conducted on the best hardware configurations found during the global search phase to converge on one or a few hardware configurations and software mappings that are optimal with respect to the entire network.

3.5 Special Operators

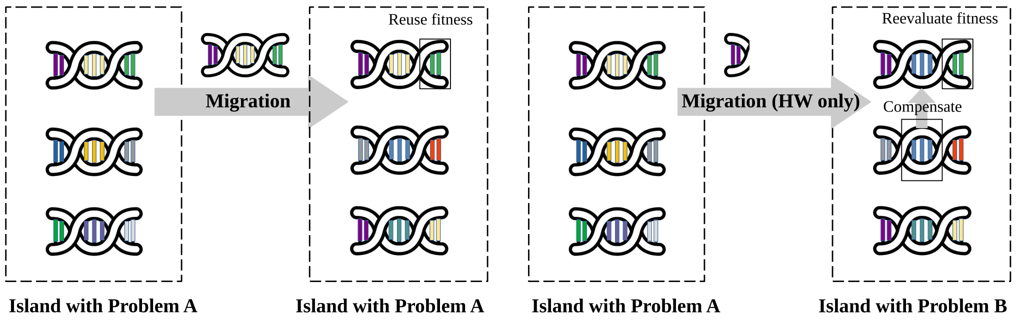
PIM allows the evolution of multiple islands in parallel and improves the quality of the solution by interleaving exploration and exploitation. In our approach, a multi-objective genetic algorithm was chosen as the evolution engine. Figure 8 illustrates the encoding of the design parameters and the supported operators. Figure 8(a) shows the encoding of the hardware and software mapspace choices to the chromosomes of a local population. Figure 8(b) illustrates crossover and mutation operators within a local population. These operators depend on three parameters: the probability of crossover, the probability of mutation, and the size of the population. Figure 8(c) and (d) illustrates the two possible migration scenarios. For migrations between islands with the same



(a) Gene encoding



(b) Crossover and mutation



(c) Migration on islands with homogeneous problems (d) Migration on islands with heterogeneous problems

Fig. 8. Gene encoding and special operators.

software mapspaces, chromosomes are compatible and can be migrated and re-used directly as shown in Figure 8(c). However, when migration occurs between the heterogeneous islands with different software parameter spaces, the chromosomes of the migrants are incompatible. As shown in Figure 8(d), we migrate only the hardware mapping and discard the software mapping since the hardware genes are globally compatible across the islands. The missing genes are compensated by copying the parameters of an existing configuration from the local population of the destination island.

To summarize, SENNA's migration algorithm performs the following steps for each new generation:

(1) Choose random source and destination islands and select migrants based on elitism.
(2) Check if the source and destination island have compatible software mapspaces.
(3) If yes, the software encoding of all migrants is kept. Go to step 6.
(4) Otherwise, the software mapping of the migrants are initialized with mappings from members of the destination island's population.
(5) Reevaluate the fitness of the migrants' new chromosomes on the destination island.
(6) Merge the migrants with the population and update the population based on elitism.
(7) Proceed to the evolution phase.

4 Evaluation

This section evaluates SENNA with three parametrizable architectures and four neural networks. We begin with the analysis of SENNA's performance with respect to the recommended hardware configuration for a given network. We then present the evaluation of Pareto-approximated solutions and discuss insights from the design while comparing SENNA to an Oracle scheme and the state-of-the-art, DiGamma [26].

Table 2. Available Design Choices of the Architectures

Architectures	HW components	Accelerator Design Choices
Eyeriss	PEs	14,28,42,56,70,84,98,112,126,140,154,168,182, 196,210,224,238,252,266,280,294,308,322,336
	Buffers (kB)	4,8,12,16,20,24,28,32
DianNao	PEs	256,288,320,352,384,416,448
	Buffers (B)	256,384,512,640,768,896,1024,1152 1280,1408,1536,1664,1792,1920,2048
Simba	PEs	2,4,6,8,10,12,14,16,18,20,22,24,26,28,30,32
	Buffers (B)	1024,1536,2048,2560,3072,3584,4096

4.1 Evaluation Environment

The search metrics such as performance, energy, and die area are computed with two parametrizable architecture simulation environments built into two independent open-source accelerator infrastructures, Timeloop [39] and Maestro [29]. These frameworks model the execution of a neural network on a parametrizable accelerator hardware instance and extract several metrics. Both frameworks employ microarchitectural models to obtain the energy requirements, the total execution cycles, the hardware utilization, and the die area. Timeloop and Maestro differ significantly with respect to the supported architecture descriptions, the search space encoding, and the design constraints. SENNA can work with both environments as a backend to extract the performance metrics of a hardware/software configuration.

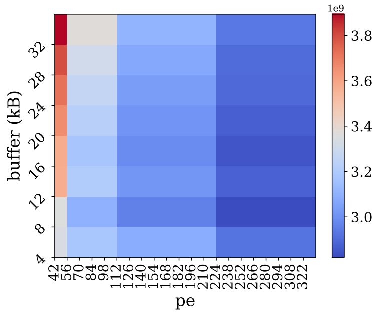
SENNA and the baseline approaches are evaluated with three different parametrizable hardware platforms and four neural networks. The hardware platforms include Eyeriss [11], DianNao [8], and Simba [44]. Table 2 lists the parameter space for the architectures. All architectures are parametrizable with respect to the number of PEs and the on-chip buffer size. The evaluated neural networks are ResNet50 (53 layers) [21], VGG16 (13 layers) [46], MobileNet-V2 (52 layers) [43], and MnasNet (52 layers) [50].

The presented evolutionary PIM-based algorithm is implemented in Pagmo [4], an open-source library for parallel optimization. To support the techniques discussed in this work, we extend Pagmo as follows. First, we have modified the multi-objective optimization solver to support constraints and deal with infeasible encodings. Second, we have improved the implementation of the PIM to support heterogeneous islands and tailored migration.

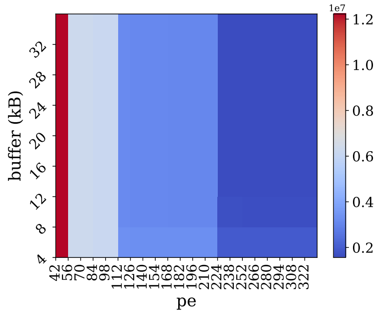
4.2 Search Algorithm and PIM Configurations

The search is guided by the evolutionary **Non-dominated Sorting Genetic Algorithm (NSGA-II)** [16], a representative and widely used multi-objective algorithm. A neural network is mapped into a PIM as follows: In the first step, we isolate all unique layers with respect to their operators and parameters. For each unique layer, four islands are allocated to increase the population diversity and minimize the likelihood of getting stuck in local minima. Each island is initialized with a population size of 100 chromosomes. The search is conducted for a total of 100 generations with a crossover probability of 95% and a mutation probability of 70%. The mutation probability is relatively high compared to conventional genetic algorithms because the design space contains many invalid configurations.

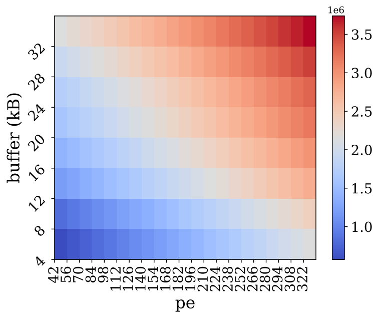
The evolutionary algorithm runs in two phases, the global search and the fine-tuning phase (Section 3.1). When the first phase completes, the best configurations for each layer are combined and form the initial population of the fine-tuning phase. In this evaluation, each island in the global search phase is initially seeded with 100 chromosomes. After an evolution over 50 generations,



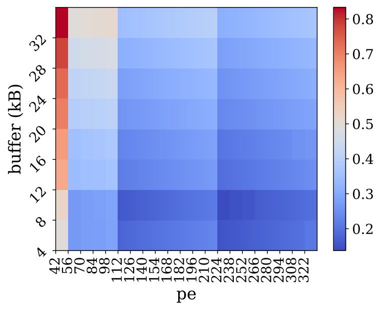
(a) Energy



(b) Cycles



(c) Die Area



(d) Overall

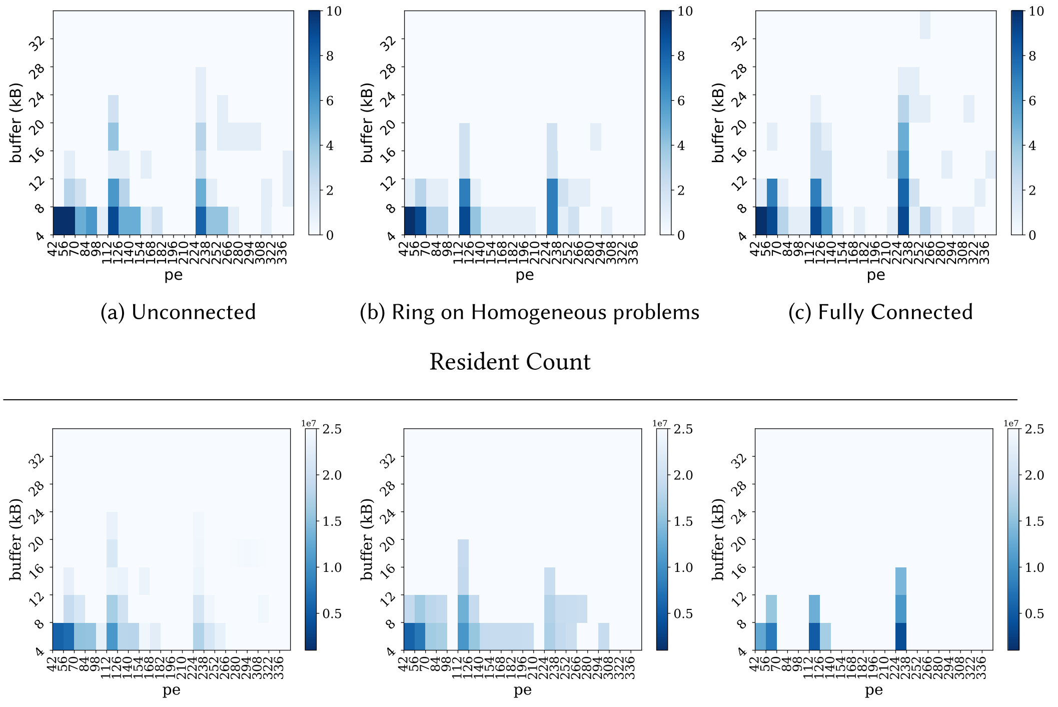
Fig. 9. Exhaustive evaluation of performance metrics of MobileNet-V2 on the Eyeriss architecture. Figures (a), (b), and (c) visualize the heatmap for the individual metrics energy, latency (cycles), and die area. Figure (d) shows the average of the individual metrics normalized to 1. Blue is better.

the best 5 candidates of each layer are selected to seed the initial population of the fine-tuning phase which evolves for another 50 generations. These values have empirically been found to work reasonably well with the employed parametrizable hardware architectures and networks but may need to be tuned to different environments.

4.3 Architecture Design Space Search Validation

We validate the architecture design search by analyzing and comparing whether the identified hardware design points match the expected best hardware designs. We demonstrate SENNA's hardware recommendation for MobileNet-V2 [43] on the Eyeriss architecture [11]. The objectives are visualized using heatmaps in this two-dimensional parameter space. The ground truth is collected by an initial exhaustive search through all possible parameter configurations. Figure 9(a), (b), and (c) shows the performance with respect to a single metric energy, latency (cycles), and chip die area.

We observe that the optimal hardware design point differs according to the target metric. For example, the design point (224 PEs, 8 kB buffer) shows the best value for the metrics energy and cycles. However, this configuration does not dominate the configuration (112 PEs, 8 kB buffer) since it requires a significantly larger die area. Figure 9(d) visualizes the quality of each hardware design point in consideration of competing objectives. To enhance the interpretability, the objectives are linearly scaled to the range [0,1) before visualization and aggregated using the average of all three metrics [37]. The heatmap clearly shows the design points for which all three metrics achieve good values and are suitable to represent the tradeoffs among the objectives. The selection of the final architecture configuration can be assisted by weighing the individual metrics differently.



(a) Unconnected (b) Ring on Homogeneous problems (c) Fully Connected

Resident Count

(d) Unconnected (e) Ring on Homogeneous problems (f) Fully Connected

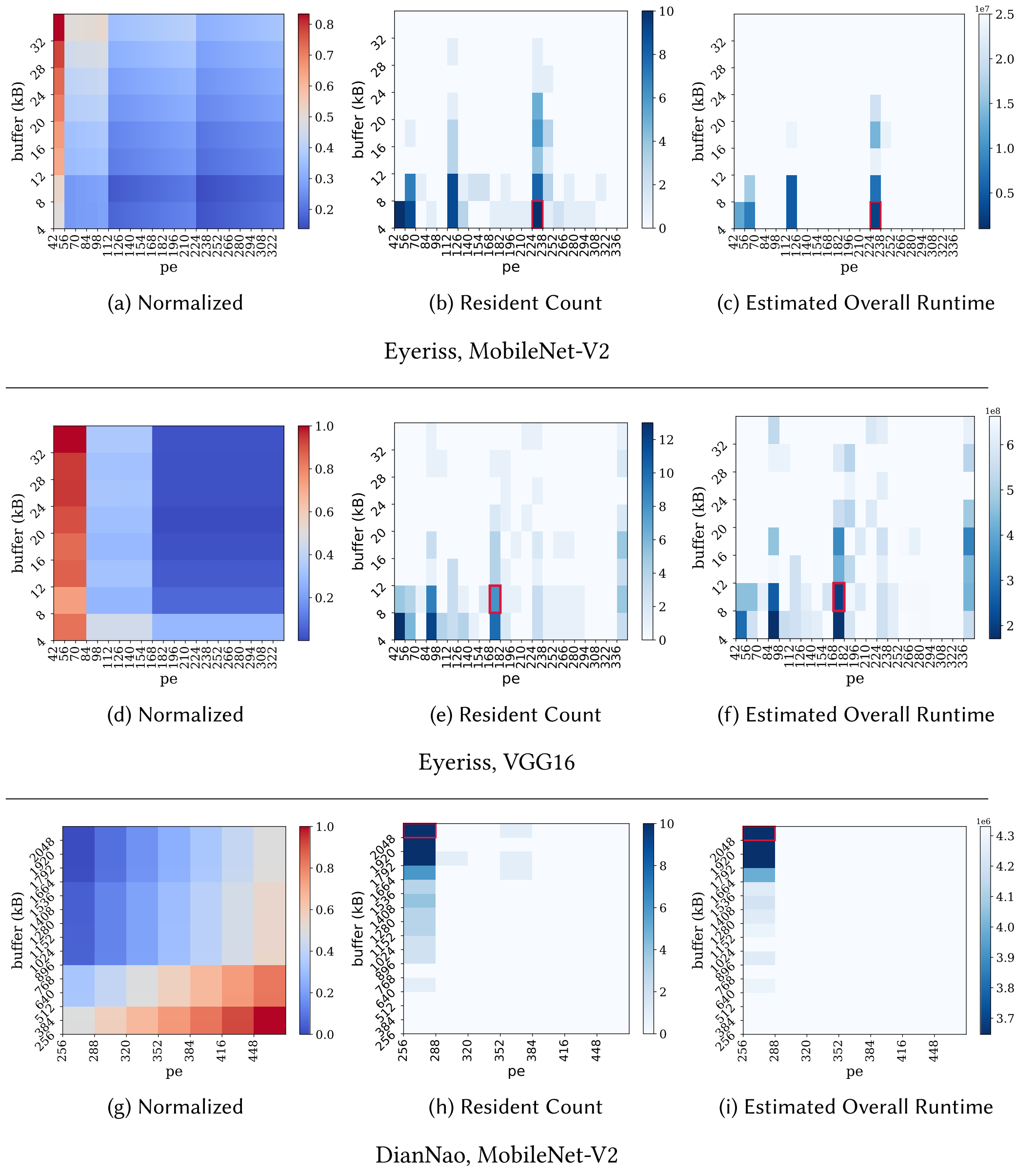
Estimated Overall Runtime

Fig. 10. Effect of the PIM island connectivity on the identified best hardware configurations.

4.4 PIM Topology

Next, we evaluate the impact of different interconnection topologies of the PIM on the recommended hardware and the runtime of the search algorithm. The results are shown in Figure 10. The upper row shows the resident count, that is, the number of hardware configurations found in the Pareto-approximated sets of unique layers. The darker a point, the more often this hardware configuration appeared in the Pareto-approximated sets. The more frequently a hardware configuration appears in the Pareto-approximated set, the higher the likelihood that this configuration yields higher performance. The lower row visualizes the estimated overall runtime. The desired result is a heatmap that shows the darkest blue for the configurations that have been identified as optimal through an exhaustive search in Figure 9(d).

Figure 10(a) and (d) shows the results for a PIM with unconnected islands, i.e., there is no migration between islands. We observe that there is little convergence on the best hardware configuration not only after the global search—Figure 10(a)—thus, results in inaccurate estimation of overall runtime in Figure 10(d). This is an expected result since each island individually optimizes only for its layer parameters. In Figure 10(b) and (e), islands with identical parameters are connected with a ring, which is the topology used for the fine-tuning phase as shown in Figure 5. This allows the exchange of full chromosomes and helps the connected populations to escape from local minima and converge towards one optimal configuration per layer. Compared to the unconnected PIM, the convergence of the global search is better; however, the absence of the connection between different layers dilutes the results. Figure 10(c) and (f), finally, visualizes the result of a fully-connected PIM (Figure 5). SENNA's tailored allows the exchange of hardware configurations that are promising



(a) Normalized (b) Resident Count (c) Estimated Overall Runtime

Eyeriss, MobileNet-V2

(d) Normalized (e) Resident Count (f) Estimated Overall Runtime

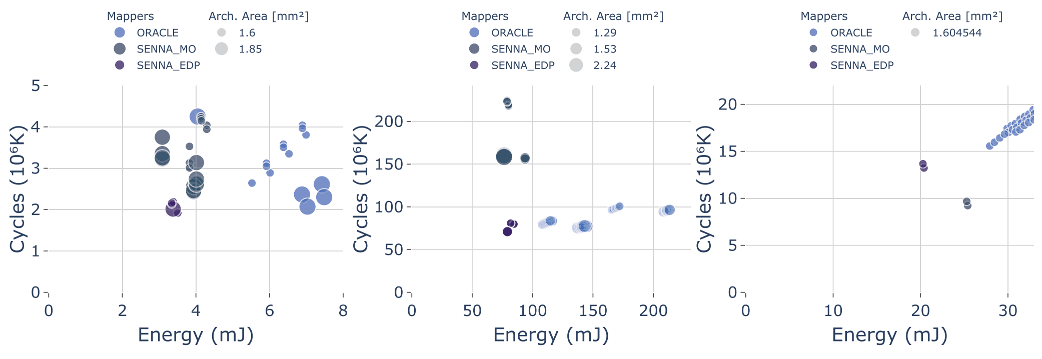
Eyeriss, VGG16

(g) Normalized (h) Resident Count (i) Estimated Overall Runtime

DianNao, MobileNet-V2

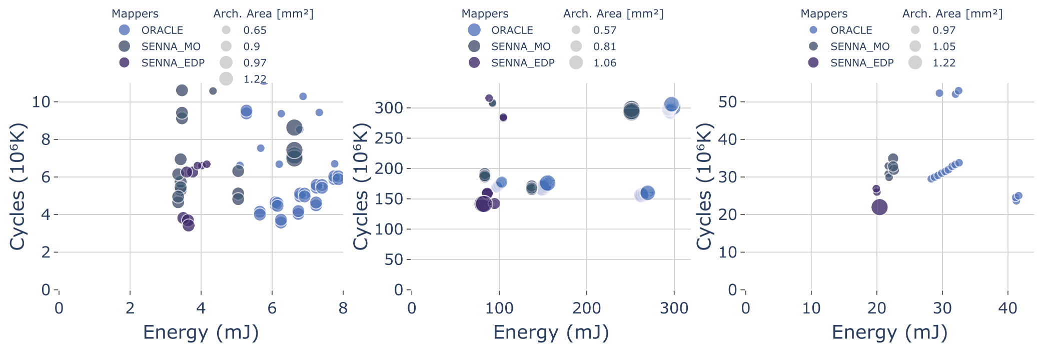
Fig. 11. Comparing the best identified architecture designs with the ideal case. Figure 11(a), (d), and (g) scales the ground truth of energy, cycles, and area into equal range and visualizes the best HW design points. Figure 11(b), (e), and (h) illustrates the number of HW design configurations found on pareto sets of individual layers (blue: all layers on the network map to the same best hardware configuration.) Figure 11(c), (f), and (i) illustrates the estimated overall runtime (blue is better.) Red box indicates the best hardware configuration found after the fine tuning phase.

with respect to a single island (i.e., layer) with all other islands and thus converge towards globally optimal configurations. We observe good convergence on the two configurations that have been identified as optimal by exhaustive search (112 and 224PEs with a 4 and 8 KB buffer). We note that while resident count shows the absence of the mapping that hinders to calculate the overall



(a) MobileNet-V2 (b) VGG16 (c) ResNet50

High Area (Area > $1.25mm^2$)



(d) MobileNet-V2 (e) VGG16 (f) ResNet50

Low Area (Area < $1.25mm^2$)

Fig. 12. Pareto front generated by SENNA_MO, SENNA_EDP, and ORACLE for different networks on Eyeriss Architecture. The size of the design point describes the area size.

performance, an approximated objectives of missing mappings enable us to closely capture the globally optimal configurations.

4.5 Hardware Recommendation

Since multiple options exist for the best hardware design, SENNA provides multiple best candidates for hardware configuration. Figure 11 compares the recommended hardware configurations with the results from the exhaustive search. Figure 11(a), (d), and (g) illustrates the quality architecture designs obtained through exhaustive search. Figure 11(b), (e), and (h) indicates the number of architectural designs discovered from the Pareto-approximated set in a layer-wise manner. It is worth to note that in Figure 11(b), and (e), the design point (42 PEs, 4 kB buffer) is found to be non-dominant on every layer though it is incapable of delivering the best performance. The rationale behind this selection is the non-dominance relation between objectives of the Pareto-approximated set, which has the lowest area despite the higher energy and runtime. The presence of these spurious solutions can misguide the multi-objective algorithm to the suboptimal design point, which reduces the chance of finding the globally optimized configurations. Hence we consider multiple design points as the best hardware configurations to embrace optimal design points and conduct the fine-tuning phase, as shown in Figure 12. This can be extended in the future by a search method

Table 3. Comparison of the Hardware Configurations Found by SENNA and the Oracle in Terms of Energy Consumption, Die Area, and Performance

	Network	MobileNet			VGG16			ResNet50		
	Architecture	Eyeriss	Diannao	Simba	Eyeriss	Diannao	Simba	Eyeriss	Diannao	Simba
Energy (mJ)	`SENNA_MO`	3.93	15.15	5.71	76.37	1,034.1	169.92	21.86	118.45	19.13
	`SENNA_EDP`	3.38	15.15	4.19	79.14	225.3	103.46	20.39	80.3	25.03
	`ORACLE`	5.52	15.13	15.73	108.26	2,177.3	NaN	27.91	423.43	NaN
Area (mm^2)	`SENNA_MO`	1.85	0.37	1.76	1.53	0.37	1.73	1.22	0.37	1.83
	`SENNA_EDP`	1.61	0.37	3.09	1.53	0.37	2.23	1.22	0.37	1.87
	`ORACLE`	1.85	0.37	1.14	1.53	0.37	NaN	1.22	0.37	NaN
Cycles (10^4K)	`SENNA_MO`	2.43	1.69	15.58	158.74	48.5	547.36	29.91	10.89	97.7
	`SENNA_EDP`	2.01	1.69	10.08	70.77	48.22	411.58	13.22	10.85	82.45
	`ORACLE`	2.64	1.69	31.9	79.56	47.99	NaN	15.91	10.82	NaN
Time (hrs)	`SENNA_MO` `SENNA_EDP`	4	6	9	7	9	9	20	18	27
	`ORACLE`	14	15	84	27	43	74	47	45	175
Budget (10^4)	`SENNA_MO` `SENNA_EDP`	53.87	56.49	67.51	70.16	74.23	106.20	147.78	153.26	192.47
	`ORACLE`	54.67	57.28	68.31	71.20	75.27	107.24	150.02	155.51	194.71

The last columns show the search time of the two frameworks. The Timeloop mapper-based Oracle often fails to find valid configurations for layers on the Simba architecture because of the large ratio of invalid configurations.

to be vigilant on suboptimal configurations and remediate. Figure 11(c), (f), and (i) illustrates the estimated overall runtime. We observe that the recommendation of the multiple hardware configurations based on the estimated overall runtime embraces globally optimal design points.

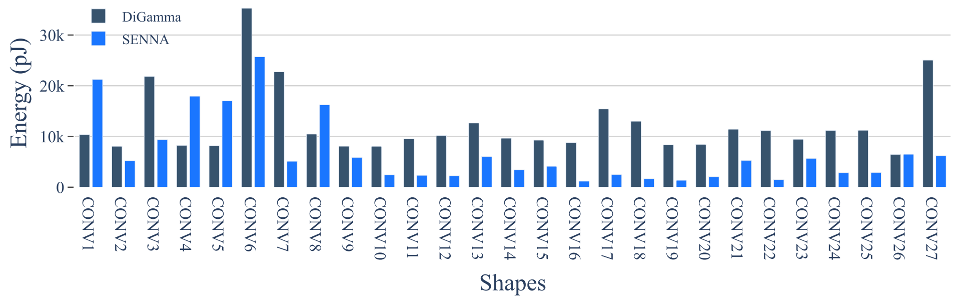
4.6 Comparison to an Oracle and Related Work

Let us demonstrate the benefit of SENNA's joint hardware/software configuration search compared to an Oracle scheme and to State-of-the-Art techniques.

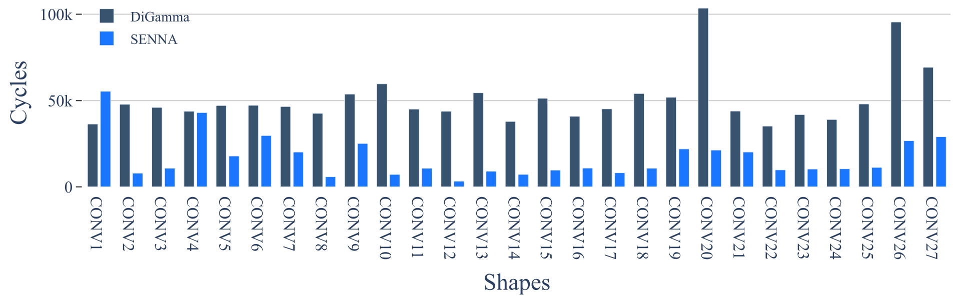
4.6.1 Comparison to Oracle Scheme. We evaluate SENNA with respect to the Oracle scheme which always selects the best hardware configuration and then consumes the entire budget on the software mapspace search on the selected hardware configuration. The rationale behind this scheme is to design a computationally viable approach that replaces exhaustive search with little optimality loss. With the extremely large design space, an exhaustive search is infeasible and time-consuming. To enable the comparison with little optimality loss, we design the oracle scheme which finds the best hardware configurations through iterative search before exploration.

To fully examine the hardware configurations, we start from the size of the budget that default configuration consumes. Then, it is executed iteratively with doubled budget until the overall performance converges. Given the best 10% of hardware configurations, the oracle scheme distributes the evaluation budget and focuses to search for the best possible software mapping. Oracle scheme employs the Timeloop-mapper as a software mapspace search engine, since other State-of-the-Arts does not support the exploration on a baseline architectures. Since Timeloop-mapper shows the best result on the **energy-delay product (EDP)** metric, SENNA also supports the EDP as an optimization goal for fair comparison. In the following, we denote the three-objective version of SENNA (Energy, Area, Cycles) with `SENNA_MO` and the two-objective version (EDP, Area) with `SENNA_EDP`.

Table 3 compares the results of `SENNA_MO`, `SENNA_EDP` with `ORACLE`. We made two main observations. First, the overall energy and runtime performance of SENNA achieves a Pareto-efficient



(a) Energy results



(b) Runtime results

Fig. 13. Comparing SENNA to DiGamma for 2D convolutions on MobileNet-V2.

solution which dominates or is in non-dominance relation compared to the ORACLE. Despite ORACLE achieves more budget to run a mapspace search on the best HW design, SENNA is capable of finding the best architecture design and its corresponding mappings. Second, the exploration in huge design spaces with many infeasible design points allows us to verify the scalability of SENNA. Simba has a search space of $O(10^{24})$ filled with invalid configurations. SENNA produces valid mappings across the network and shows robustness in huge design spaces, while ORACLE fails to generate mappings layer by layer. Figure 12 shows the distribution of overall energy and runtime performance with Pareto Front on different networks. With respect to the solutions of Timeloop, SENNA is capable of finding the best architecture design in the limited budget of evaluations. SENNA_EDP tends to perform well compared to the SENNA_MO, since SENNA_MO optimizes on more metrics and includes more spurious solutions which dilute the result.

4.6.2 Comparison to the State-of-the-Art. To show the area and overall performance efficiency, We compare SENNA_MO with DiGamma [26]. DiGamma requires the constraints on encoding to respect the workload dimensions, by revising the loop tiling size and bounds under the size of allocated HW components. To address the complexity of the encoding, SENNA was implemented to collect the possible encoding set and apply its own encoding scheme. Since the enumeration of the possible choices is encoded into an integer sequentially, it removes the possibility of infeasible design points being chosen. Moreover, DiGamma optimizes each layer and suggests its L1 and L2 buffer requirements. We extended DiGamma to derive the maximal architecture as a final architecture design by applying the maximum size of the buffer requirements from the found solutions.

Figure 13(a) and (b) shows the found solution by plotting layers with different dimensions as the x-axis. DiGamma derives the hardware configuration of (976 PEs, 510 kB L1 buffer, 105 MB L2 buffer) with an area size of $65.75mm^2$ while SENNA found the hardware configuration of (872 PEs, 54 kB L1 buffer, 113 MB L2 buffer) with an area size of $35.80mm^2$. Overall, SENNA

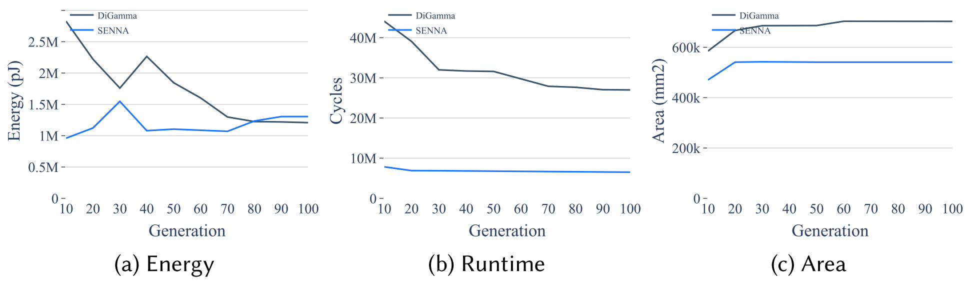


Fig. 14. Overall performance improvements of ResNet50 in terms of energy, runtime, and area.

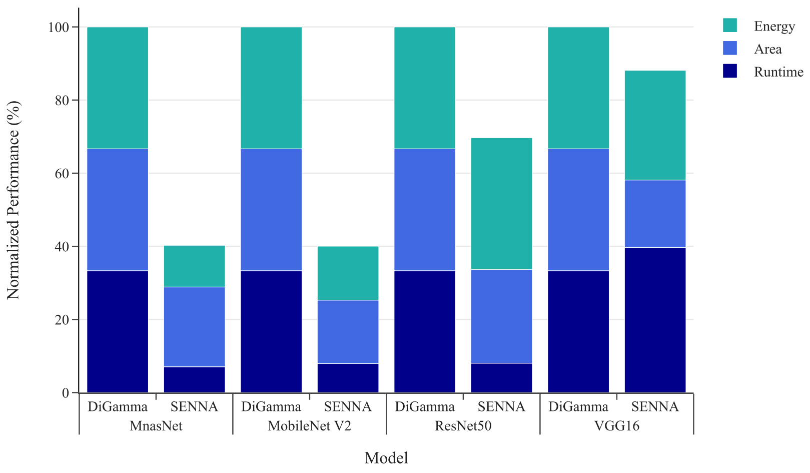


Fig. 15. Normalized overall performance of different DNNs on Maestro.

outperforms the DiGamma both in terms of energy and runtime. In particular, the solution found by SENNA occupies a 48% smaller area than the configuration of DiGamma. This is possible since the formulation of the PIM succeeds in segregating the mapspace from the architecture design space and preserving the superior mappings. Figure 14 compares the found solution of SENNA and DiGamma on multiple objectives per generation. The improvement is three-fold in runtime with minimal degradation in energy and area.

Figure 15 compares the normalized overall performance of different DNNs. Multiple normalized objectives are stacked in the direction of the Y-axis, and SENNA is normalized to the solution of DiGamma. We found that this work benefits in improving the overall performance of deeper and wider networks. The solution outperforms the quality of DiGamma as the network goes deeper with a higher duplication of identical layers. For example, VGG16 has nine unique layers of 13 layers. It has less duplication on layers with low reusability on found solutions and achieves relatively less improvement. On the other hand, deeper networks achieve decent performance improvement by obtaining more chances of reusing the found solutions. ResNet50 has 23 unique layers of 53, MobileNet-V2 has 27 unique layers of 52, and MnasNet has 33 unique layers of 52, respectively. To summarize, PIM finds better solutions not only for each layer but also for a given network.

5 Conclusion

While hardware/software co-design has been widely studied for parametrizable neural network accelerators, finding a practical design that considers the end-to-end network performance under multiple objectives is still an open question. This work presents the SENNA framework that employs a PIM, which unifies the mapspace to represent a network and optimizes for a given accelerator design in a single execution. We harness the unified design space and multi-objective search to significantly improve the efficiency of the search. A comparison with state-of-the-art mapper shows that SENNA is able to find better overall configurations in a significantly shorter time.

References

[1] Inpyo Bae, Barend Harris, Hyemi Min, and Bernhard Egger. 2018. Auto-tuning CNNs for coarse-grained reconfigurable array-based accelerators. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 37, 10 (2018), 1–1. DOI: https://doi.org/10.1109/TCAD.2018.2857278

[2] Riyadh Baghdadi, Jessica Ray, Malek Ben Romdhane, Emanuele Del Sozzo, Abdurrahman Akkas, Yunming Zhang, Patricia Suriana, Shoaib Kamil, and Saman Amarasinghe. 2019. Tiramisu: A polyhedral compiler for expressing fast and portable code. In *Proceedings of the 2019 IEEE/ACM International Symposium on Code Generation and Optimization (CGO 2019)*. IEEE, 193–205.

[3] Slim Bechikh, Lamjed Ben Said, and Khaled Ghedira. 2010. Estimating nadir point in multi-objective optimization using mobile reference points. In *Proceedings of the IEEE Congress on Evolutionary Computation*. 1–9. DOI: https://doi.org/10.1109/CEC.2010.5586203

[4] Francesco Biscani and Dario Izzo. 2020. A parallel global multiobjective framework for optimization: pagmo. *Journal of Open Source Software* 5, 53 (2020), 2338. DOI: https://doi.org/10.21105/joss.02338

[5] Christian Blum and Andrea Roli. 2003. Metaheuristics in combinatorial optimization: Overview and conceptual comparison. *ACM Computing Surveys* 35, 3 (2003), 268–308. DOI: https://doi.org/10.1145/937503.937505

[6] Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. 2020. Language models are few-shot learners. In *Proceedings of the 34th International Conference on Neural Information Processing Systems (NIPS'20)*. Curran Associates Inc., Red Hook, NY, USA, Article 159, 25 pages.

[7] Liang-Chieh Chen, George Papandreou, Iasonas Kokkinos, Kevin Murphy, and Alan L. Yuille. 2018. DeepLab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected CRFs. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 40, 4 (2018), 834–848. DOI: https://doi.org/10.1109/TPAMI.2017.2699184

[8] Tianshi Chen, Zidong Du, Ninghui Sun, Jia Wang, Chengyong Wu, Yunji Chen, and Olivier Temam. 2014. DianNao: A small-footprint high-throughput accelerator for ubiquitous machine-learning. *ACM SIGARCH Computer Architecture News* 42, 1 (2014), 269–284. DOI: https://doi.org/10.1145/2654822.2541967g

[9] Tianqi Chen, Thierry Moreau, Ziheng Jiang, Lianmin Zheng, Eddie Yan, Meghan Cowan, Haichen Shen, Leyuan Wang, Yuwei Hu, Luis Ceze, Carlos Guestrin, and Arvind Krishnamurthy. 2018. TVM: An automated end-to-end optimizing compiler for deep learning. In *Proceedings of the 13th USENIX Conference on Operating Systems Design and Implementation (OSDI'18)*. USENIX Association, USA, 579–594.

[10] Yiran Chen, Yuan Xie, Linghao Song, Fan Chen, and Tianqi Tang. 2020. A survey of accelerator architectures for deep neural networks. *Engineering* 6, 3 (2020), 264–274. DOI: https://doi.org/10.1016/j.eng.2020.01.007

[11] Yu-Hsin Chen, Joel Emer, and Vivienne Sze. 2016. Eyeriss: A spatial architecture for energy-efficient dataflow for convolutional neural networks. In *Proceedings of the 43rd International Symposium on Computer Architecture (ISCA'16)*. IEEE, 367–379. DOI: https://doi.org/10.1109/ISCA.2016.40

[12] Matthieu Courbariaux, Yoshua Bengio, and Jean-Pierre David. 2014. Training deep neural networks with low precision multiplications. arXiv e-prints (Dec. 2014). https://doi.org/10.48550/arXiv.1412.7024

[13] Teodor Gabriel Crainic and Michel Toulouse. 2003. *Parallel Strategies for Meta-Heuristics*. Springer US, Boston, MA, 475–513. DOI: https://doi.org/10.1007/0-306-48056-5_17

[14] Shail Dave, Youngbin Kim, Sasikanth Avancha, Kyoungwoo Lee, and Aviral Shrivastava. 2019. DMazeRunner: Executing perfectly nested loops on dataflow accelerators. *ACM Transactions on Embedded Computing Systems* 18, 5s, Article 70 (2019), 27 pages. DOI: https://doi.org/10.1145/3358198

[15] Kalyanmoy Deb, Shamik Chaudhuri, and Kaisa Miettinen. 2006. Towards estimating nadir objective vector using evolutionary approaches. In *Proceedings of the 8th Annual Conference on Genetic and Evolutionary Computation (GECCO'06)*. Association for Computing Machinery, New York, NY, USA, 643–650. DOI: https://doi.org/10.1145/1143997.1144113

[16] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan. 2002. A fast and elitist multiobjective genetic algorithm: NSGA-II. *IEEE Transactions on Evolutionary Computation* 6, 2 (2002), 182–197. DOI: https://doi.org/10.1109/4235.996017

[17] Li Du, Yuan Du, Yilei Li, Junjie Su, Yen-Cheng Kuan, Chun-Chen Liu, and Mau-Chung Frank Chang. 2017. A reconfigurable streaming deep convolutional neural network accelerator for Internet of Things. *IEEE Transactions on Circuits and Systems I: Regular Papers* 65, 1 (2017), 198–208.

[18] Zidong Du, Robert Fasthuber, Tianshi Chen, Paolo Ienne, Ling Li, Tao Luo, Xiaobing Feng, Yunji Chen, and Olivier Temam. 2015. ShiDianNao: Shifting vision processing closer to the sensor. In *Proceedings of the 2015 ACM/IEEE 42nd Annual International Symposium on Computer Architecture (ISCA)*. 92–104. DOI: https://doi.org/10.1145/2749469.2750389

[19] Yonggan Fu, Yongan Zhang, Yang Zhang, David Cox, and Yingyan Lin. 2021. Auto-NBA: Efficient and effective search over the joint space of networks, bitwidths, and accelerators. In *Proceedings of the 38th International Conference on Machine Learning.* Marina Meila and Tong Zhang (Eds.), Proceedings of Machine Learning Research, Vol. 139, PMLR, 3505–3517. Retrieved from https://proceedings.mlr.press/v139/fu21d.html

[20] Mingyu Gao, Jing Pu, Xuan Yang, Mark Horowitz, and Christos Kozyrakis. 2017. TETRIS: Scalable and efficient neural network acceleration with 3D memory. In *Proceedings of the 22nd International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS'17)*. Association for Computing Machinery, New York, NY, USA, 751–764. DOI: https://doi.org/10.1145/3037697.3037702

[21] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

[22] Kartik Hegde, Po-An Tsai, Sitao Huang, Vikas Chandra, Angshuman Parashar, and Christopher W. Fletcher. 2021. 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 (ASPLOS'21)*. Association for Computing Machinery, New York, NY, USA, 943–958. DOI: https://doi.org/10.1145/3445814.3446762

[23] Qijing Huang, Minwoo Kang, Grace Dinh, Thomas Norell, Aravind Kalaiah, James Demmel, John Wawrzynek, and Yakun Sophia Shao. 2021. CoSA: Scheduling by constrained optimization for spatial accelerators. In *Proceedings of the 48th Annual International Symposium on Computer Architecture (ISCA'21)*. IEEE, 554–566. DOI: https://doi.org/10.1109/ISCA52012.2021.00050

[24] Sheng-Chun Kao, Geonhwa Jeong, and Tushar Krishna. 2020. ConfuciuX: Autonomous hardware resource assignment for DNN accelerators using reinforcement learning. In *Proceedings of the 53rd Annual IEEE/ACM International Symposium on Microarchitecture, MICRO*. IEEE, 622–636.

[25] Sheng-Chun Kao and Tushar Krishna. 2020. GAMMA: Automating the HW mapping of DNN models on accelerators via genetic algorithm. In *Proceedings of the 39th International Conference on Computer-Aided Design (ICCAD'20)*. Article 44, 9 pages. DOI: https://doi.org/10.1145/3400302.3415639

[26] Sheng-Chun Kao, Michael Pellauer, Angshuman Parashar, and Tushar Krishna. 2022. DiGamma: Domain-aware genetic algorithm for HW-mapping Co-optimization for DNN accelerators. In *Proceedings of the 2022 Conference and Exhibition on Design, Automation and Test in Europe (DATE'22)*. European Design and Automation Association, Leuven, BEL, 232–237.

[27] Daniel Martin Katz, Michael James Bommarito, Shang Gao, and Pablo Arredondo. 2024. Gpt-4 passes the bar exam. *Philosophical Transactions of the Royal Society A* 382, 2270 (2024), 20230254.

[28] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. 2017. ImageNet classification with deep convolutional neural networks. *Communications of the ACM* 60, 6 (2017), 84–90. DOI: https://doi.org/10.1145/3065386

[29] Hyoukjun Kwon, Prasanth Chatarasi, Vivek Sarkar, Tushar Krishna, Michael Pellauer, and Angshuman Parashar. 2020. MAESTRO: A data-centric approach to understand reuse, performance, and hardware cost of DNN mappings. *IEEE Micro* 40, 3 (2020), 20–29. DOI: https://doi.org/10.1109/MM.2020.2985963

[30] Yuhong Li, Cong Hao, Xiaofan Zhang, Xinheng Liu, Yao Chen, Jinjun Xiong, Wen-mei Hwu, and Deming Chen. 2020. EDD: Efficient differentiable DNN architecture and implementation co-search for embedded AI solutions. In *Proceedings of the 2020 57th ACM/IEEE Design Automation Conference (DAC)*. 1–6. DOI: https://doi.org/10.1109/DAC18072.2020.9218749

[31] Jonathan Long, Evan Shelhamer, and Trevor Darrell. 2015. Fully convolutional networks for semantic segmentation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

[32] Tao Luo, Shaoli Liu, Ling Li, Yuqing Wang, Shijin Zhang, Tianshi Chen, Zhiwei Xu, Olivier Temam, and Yunji Chen. 2017. DaDianNao: A neural network supercomputer. *IEEE Transactions on Computers* 66, 1 (2017), 73–88. DOI: https://doi.org/10.1109/TC.2016.2574353

[33] Hyemi Min, Jungyoon Kwon, and Bernhard Egger. 2021. Fast generation of optimized execution plans for parameterizable CNN accelerators: Work-in-progress. In *Proceedings of the 2021 International Conference on Compilers, Architectures, and Synthesis for Embedded Systems (CASES'21)*. Association for Computing Machinery, New York, NY, USA, 1–2. DOI: https://doi.org/10.1145/3451939.3477593

[34] Hyemi Min, Jungyoon Kwon, and Bernhard Egger. 2023. Flexer: Out-of-order scheduling for multi-NPUs. In *Proceedings of the 2023 International Symposium on Code Generation and Optimization (CGO'23).* ACM, New York, NY, USA, 12 pages. DOI: https://doi.org/10.1145/3579990.3580025

[35] Pavlo Molchanov, Stephen Tyree, Tero Karras, Timo Aila, and Jan Kautz. 2016. Pruning Convolutional Neural Networks for Resource Efficient Transfer Learning. *CoRR* (2016).

[36] Luigi Nardi, Artur Souza, David Koeplinger, and Kunle Olukotun. 2019. HyperMapper: A practical design space exploration framework. In *Proceedings of the 2019 IEEE 27th International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS).* 425–426. DOI: https://doi.org/10.1109/MASCOTS.2019.00053

[37] Alireza Nazemi, Andrew H. Chan, and Xin Yao. 2008. Selecting representative parameters of rainfall-runoff models using multi-objective calibration results and a fuzzy clustering algorithm. In *Proceedings of the BHS 10th National Hydrology Symposium, Exeter, UK.* 13–20.

[38] Kristian Robert Nichols. 2003. *A Reconfigurable Computing Architecture for Implementing Artificial Neural Networks on FPGA.* Ph. D. Dissertation. University of Guelph.

[39] Angshuman Parashar, Priyanka Raina, Yakun Sophia Shao, Yu-Hsin Chen, Victor A. Ying, Anurag Mukkara, Rangharajan Venkatesan, Brucek Khailany, Stephen W. Keckler, and Joel Emer. 2019. Timeloop: A systematic approach to DNN accelerator evaluation. In *Proceedings of the 2019 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS).* 304–315. DOI: https://doi.org/10.1109/ISPASS.2019.00042

[40] Mario Porrmann, Ulf Witkowski, Heiko Kalte, and Ulrich Ruckert. 2002. Implementation of artificial neural networks on a reconfigurable hardware accelerator. In *Proceedings of the 10th Euromicro Workshop on Parallel, Distributed and Network-based Processing.* IEEE, 243–250.

[41] Enrico Russo, Maurizio Palesi, Salvatore Monteleone, Davide Patti, Giuseppe Ascia, and Vincenzo Catania. 2022. MEDEA: A multi-objective evolutionary approach to DNN hardware mapping. In *Proceedings of the 2022 Design, Automation and Test in Europe Conference and Exhibition (DATE).* 226–231. DOI: https://doi.org/10.23919/DATE54114.2022.9774747

[42] Enrico Russo, Maurizio Palesi, Davide Patti, Salvatore Monteleone, Giuseppe Ascia, and Vincenzo Catania. 2023. Multiobjective end-to-end design space exploration of parameterized DNN accelerators. *IEEE Internet of Things Journal* 10, 2 (2023), 1800–1812. DOI: https://doi.org/10.1109/JIOT.2022.3209401

[43] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. 2018. MobileNetV2: Inverted residuals and linear bottlenecks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR).*

[44] Yakun Sophia Shao, Jason Clemons, Rangharajan Venkatesan, Brian Zimmer, Matthew Fojtik, Nan Jiang, Ben Keller, Alicia Klinefelter, Nathaniel Pinckney, Priyanka Raina, Stephen G. Tell, Yanqing Zhang, William J. Dally, Joel Emer, C. Thomas Gray, Brucek Khailany, and Stephen W. Keckler. 2019. Simba: Scaling deep-learning inference with multi-chip-module-based architecture. In *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO'52).* Association for Computing Machinery, New York, NY, USA, 14–27. DOI: https://doi.org/10.1145/3352460.3358302

[45] David Silver, Aja Huang, Chris J. Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. 2016. Mastering the game of Go with deep neural networks and tree search. *Nature* 529, 7587 (2016), 484–489.

[46] Karen Simonyan and Andrew Zisserman. 2015. Very deep convolutional networks for large-scale image recognition. In *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015.* Conference Track Proceedings.

[47] Zbigniew Skolicki. 2005. An analysis of island models in evolutionary computation. In *Proceedings of the 7th Annual Workshop on Genetic and Evolutionary Computation (GECCO'05).* Association for Computing Machinery, New York, NY, USA, 386–389. DOI: https://doi.org/10.1145/1102256.1102343

[48] Morgan Stuart and Milos Manic. 2017. Survey of progress in deep neural networks for resource-constrained applications. In *Proceedings of the IECON 2017-43rd Annual Conference of the IEEE Industrial Electronics Society.* IEEE, 7259–7266.

[49] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. 2015. Going deeper with convolutions. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR).*

[50] Mingxing Tan, Bo Chen, Ruoming Pang, Vijay Vasudevan, Mark Sandler, Andrew Howard, and Quoc V. Le. 2019. MnasNet: Platform-aware neural architecture search for mobile. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR).*

[51] Miheer Vaidya, Aravind Sukumaran-Rajam, Atanas Rountev, and P. Sadayappan. 2022. Comprehensive accelerator-dataflow co-design optimization for convolutional neural networks. In *Proceedings of the 2022 IEEE/ACM*

International Symposium on Code Generation and Optimization (CGO). 325–335. DOI:https://doi.org/10.1109/CGO53902.2022.9741281

[52] Rangharajan Venkatesan, Yakun Sophia Shao, Miaorong Wang, Jason Clemons, Steve Dai, Matthew Fojtik, Ben Keller, Alicia Klinefelter, Nathaniel Pinckney, Priyanka Raina, Yanqing Zhang, Brian Zimmer, William J. Dally, Joel Emer, Stephen W. Keckler, and Brucek Khailany. 2019. MAGNet: A modular accelerator generator for neural networks. In *Proceedings of the 2019 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. 1–8. DOI:https://doi.org/10.1109/ICCAD45719.2019.8942127

[53] Jie Wang, Licheng Guo, and Jason Cong. 2021. AutoSA: A polyhedral compiler for high-performance systolic arrays on FPGA. In *Proceedings of the 2021 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays (FPGA'21)*. Association for Computing Machinery, New York, NY, USA, 93–104. DOI:https://doi.org/10.1145/3431920.3439292

[54] Qingcheng Xiao, Size Zheng, Bingzhe Wu, Pengcheng Xu, Xuehai Qian, and Yun Liang. 2021. HASCO: Towards agile hardware and software co-design for tensor computation. In *Proceedings of the 48th Annual International Symposium on Computer Architecture (ISCA'21)*. IEEE, 1055–1068. DOI:https://doi.org/10.1109/ISCA52012.2021.00086

[55] Pengfei Xu, Xiaofan Zhang, Cong Hao, Yang Zhao, Yongan Zhang, Yue Wang, Chaojian Li, Zetong Guan, Deming Chen, and Yingyan Lin. 2020. AutoDNNchip: An automated DNN chip predictor and builder for both FPGAs and ASICs. In *Proceedings of the 2020 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays (FPGA'20)*. Association for Computing Machinery, New York, NY, USA, 40–50. DOI:https://doi.org/10.1145/3373087.3375306

[56] Dan Zhang, Safeen Huda, Ebrahim Songhori, Kartik Prabhu, Quoc Le, Anna Goldie, and Azalia Mirhoseini. 2022. A full-stack search technique for domain optimized deep learning accelerators. In *Proceedings of the 27th ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS'22)*. Association for Computing Machinery, New York, NY, USA, 27–42. DOI:https://doi.org/10.1145/3503222.3507767

[57] Qingfu Zhang and Hui Li. 2007. MOEA/D: A multiobjective evolutionary algorithm based on decomposition. *IEEE Transactions on Evolutionary Computation* 11, 6 (2007), 712–731. DOI:https://doi.org/10.1109/TEVC.2007.892759

[58] Xiaofan Zhang, Hanchen Ye, Junsong Wang, Yonghua Lin, Jinjun Xiong, Wen-mei Hwu, and Deming Chen. 2020. DNNExplorer: A Framework for modeling and exploring a novel paradigm of FPGA-Based DNN accelerator. In *Proceedings of the 39th International Conference on Computer-Aided Design (ICCAD'20)*. Association for Computing Machinery, New York, NY, USA, Article 61, 9 pages. DOI:https://doi.org/10.1145/3400302.3415609

[59] Size Zheng, Yun Liang, Shuo Wang, Renze Chen, and Kaiwen Sheng. 2020. FlexTensor: An automatic schedule exploration and optimization framework for tensor computation on heterogeneous system. In *Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS'20)*. Association for Computing Machinery, New York, NY, USA, 859–873. DOI:https://doi.org/10.1145/3373376.3378508

[60] Gaofeng Zhou, Jianyang Zhou, and Haijun Lin. 2018. Research on NVIDIA deep learning accelerator. In *Proceedings of the 2018 12th IEEE International Conference on Anti-counterfeiting, Security, and Identification (ASID)*. 192–195. DOI:https://doi.org/10.1109/ICASID.2018.8693202

[61] Eckart Ziztler, Marco Laumanns, and Lothar Thiele. 2002. SPEA2: Improving the strength Pareto evolutionary algorithm for multi objective optimization. *Evolutionary Methods for Design, Optimization, and Control* (2002), 95–100.

Received 26 December 2023; revised 4 August 2024; accepted 17 October 2024