Multi-Objective Hardware-Mapping Co-Optimisation for Multi-DNN Workloads on Chiplet-based Accelerators

Abhijit Das*, *Member, IEEE,* Enrico Russo*, *Student Member, IEEE,* and Maurizio Palesi, *Senior Member, IEEE*

Abstract—The need to efficiently execute different Deep Neural Networks (DNNs) on the same computing platform, coupled with the requirement for easy scalability, makes Multi-Chip Module (MCM)-based accelerators a preferred design choice. Such an accelerator brings together heterogeneous sub-accelerators in the form of chiplets, interconnected by a Network-on-Package (NoP). This paper addresses the challenge of selecting the most suitable sub-accelerators, configuring them, determining their optimal placement in the NoP, and mapping the layers of a predetermined set of DNNs spatially and temporally. The objective is to minimise execution time and energy consumption during parallel execution while also minimising the overall cost, specifically the silicon area, of the accelerator.

This paper presents MOHaM, a framework for multi-objective hardware-mapping co-optimisation for multi-DNN workloads on chiplet-based accelerators. MOHaM exploits a multi-objective evolutionary algorithm that has been specialised for the given problem by incorporating several customised genetic operators. MOHaM is evaluated against state-of-the-art Design Space Exploration (DSE) frameworks on different multi-DNN workload scenarios. The solutions discovered by MOHaM are Pareto optimal compared to those by the state-of-the-art. Specifically, MOHaM-generated accelerator designs can reduce latency by up to 96% and energy by up to 96.12%.

Index Terms—Deep Neural Network (DNN), Accelerator, Design Space Exploration (DSE), Hardware-Mapping Co-Optimisation.

1 Introduction

Deep Neural Network (DNN) accelerators drive the era of Domain-Specific Architectures (DSAs). From edge [1] to the cloud [2], they are ubiquitous to enable performance improvement for different applications. To meet the ever-increasing computation demand from emerging applications, bigger DNN accelerators are deployed in data centers. Existing works have either proposed monolithic chip-based designs like Google TPUv4 [2] or Multi-Chip Module (MCM)-based designs like Simba [3]. The latter is preferred as a scalable and adaptable design paradigm to combine multiple accelerators (as chiplets) for building a big multi-accelerator system. Table 1 presents the existing works on multi-accelerator systems [3][2][4][5][6][7][8][9][10][11]. Here, the second column, *Hetero-Core*, indicates the availability of heterogeneous core or chiplet integration.

A primary enabler of scalability in accelerators is the support for multi-DNN workloads, where different DNNs are executed together. This becomes a fundamental aspect in multi-accelerator systems as they could house heterogeneous accelerators supporting diverse DNNs. Naturally, the third column, *Mul-DNN*, in Table 1 indicates that most existing works support multi-DNN workloads in some form.

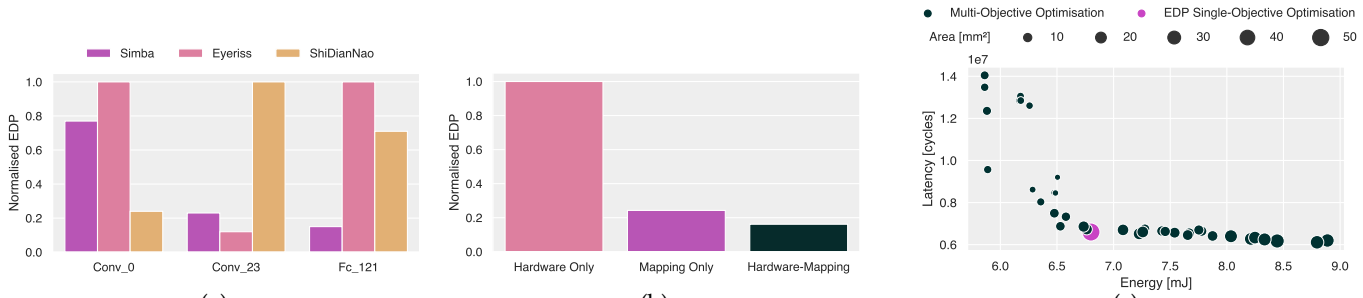
Two of the most important factors deciding a DNN accelerator's performance are its hardware configuration and mapping strategy. Their design spaces are extremely large and hence are often optimised independently [12][13][14]. With multiple heterogeneous accelerators, these design spaces are only becoming larger in multi-accelerator systems. For example, MAGMA [10] reported the design space of mapping alone to be of size $O(1e81)$. However, hardware and mapping are interdependent and, if not optimised together, may lead to sub-par performance when workload changes [15]. Furthermore, even if a multi-accelerator system is designed for known workloads, there is extreme heterogeneity in layer shapes and operations [8]. Hence, hardware-mapping co-optimisation is critical to tackle the efficiency challenges in multi-accelerator systems and improve their performance. The fourth column, *HW-MP*, in Table 1 shows the availability of hardware-mapping co-optimisation in existing multi-accelerator systems.

Table 1: Comparison of multi-accelerator systems based on the availability of heterogeneous cores (Hetero-Core), multi-DNN workloads (Mul-DNN), hardware-mapping co-optimisation (HW-MP) and multi-objective exploration (Mul-Obj). A (━) indicates unavailability of information.

Work	Hetero-Core	Mul-DNN	HW-MP	Mul-Obj
Simba [3]	✗	✗	✗	✗
TPUv4 [2]	✗	━	✗	✗
CS-2 [4]	✗	✗	✗	✗
AI-MT [5]	✓	✓	✗	✗
PREMA [6]	✓	✓	✗	✗
Planaria [7]	✗	✓	✓	✗
Herald [8]	✓	✓	✓	✗
VELTAIR [9]	✗	✓	✗	✗
MAGMA [10]	✓	✓	✗	✗
H3M [11]	✗	✓	✓	✗
MOHaM	✓	✓	✓	✓

- *A. Das is with the Department of Computer Architecture, Universitat Politécnica de Catalunya, Barcelona 08034, Spain. E-mail: abhijit.das@upc.edu*
- *E. Russo and M. Palesi are with the Department of Electrical, Electronics and Computer Engineering, University of Catania, Catania 95124, Italy. E-mail: enrico.russo@phd.unict.it; maurizio.palesi@unict.it*
- *A. Das and E. Russo are co-first authors and contributed equally.*

Manuscript received Month 00, 0000; revised Month 00, 0000.



(a) (b) (c)

Figure 1: Need for heterogeneity, co-optimisation and multi-objective exploration.

Designing an accelerator often involves multiple objectives, like latency (delay), throughput, energy, area, temperature, Total Cost of Ownership (TCO), carbon footprint, etc. A convenient way to explore is the aggregation of multiple objectives into one (mono-objective), say, Energy-Delay-Product (EDP). However, a chosen group of objectives might be conflicting with one another; in which case, aggregation does not help. With multi-accelerator systems bringing heterogeneous accelerators together, penalties for aggregation will be higher. Hence, a multi-objective exploration is necessary for identifying multiple solutions to select the most suitable one according to the requirement. Unfortunately, the last column, *Mul-Obj*, in Table 1 shows that none of the existing systems has explored the multi-objective space.

Contribution: An ideal multi-accelerator system will have all the columns ticked (✔) in Table 1, i.e., it will have *Hetero-Core* for adaptability, *Mul-DNN* for scalability, *HW-MP* for performance, and *Mul-Obj* for solution variety. However, state-of-the-art falls short in one or more of these requirements due to the unavailability of an orthogonal Design Space Exploration (DSE) framework. This work proposes such a framework for Multi-Objective Hardware-Mapping co-optimisation for multi-DNN workloads on chiplet-based accelerators (**MOHaM**)[1]. A multi-accelerator system designed using MOHaM will tick all the columns in Table 1. The proposed framework has the following features:

1) Given a multi-DNN workload and a library of heterogeneous accelerator templates, MOHaM outputs a Pareto set of multi-accelerator system designs and their associated schedules for the optimal trade-off between objectives. It is to be used at design-time.
2) MOHaM exploits the NSGA-II [16] multi-objective Genetic Algorithm (GA) and leverages the Timeloop [17] + Accelergy [18] cost model for DSE.
3) MOHaM is for designing multi-accelerator systems where the workload can be known or guessed apriori. The workload, accelerator templates and objectives can be set as per the design requirement.

This work makes the following novel contributions:

1) It frames the need for adaptability, scalability, performance and solution variety in multi-accelerator systems into an optimisation problem. It develops multiple modules to present MOHaM, the first open-source infrastructure for complete DSE of multi-accelerator systems with known workloads.
2) It designs several custom GA operators and proposes an optimisation algorithm to enable strategic exploration of the huge design space. This increases the sampling efficiency and makes MOHaM orders of magnitude faster than the exhaustive search.
3) For the evaluated state-of-the-art, MOHaM-generated system designs are capable of reducing latency by 18.40%, 13% and 96% and energy by 36.87%, 96.12% and 93.76% compared to CoSA [12], DiGamma [19] and Herald [8], respectively.

1. https://github.com/Haimrich/moham

2 Background and Motivation

2.1 Chiplet-based Accelerators

To ensure scalability, Simba [3] proposed a Multi-Chip-Module (MCM) based multi-accelerator system, where each chiplet is a DNN accelerator (called *sub-accelerator* in the bigger context) connected by a Network-on-Package (NoP). Each Sub-Accelerator (SA) has an array of Processing Elements (PEs) and a shared Global Buffer (GB) [20]. Each PE houses one or more Multiply-Accumulate (MAC) units to compute partial sums and a Local Buffer (LB) to store them. GB collects weights and activations from memory (HBM/DRAM) through the NoP and distributes them to the LBs through the Network-on-Chip (NoC). Similarly, outputs are written from LB to GB and then to memory through NoC and NoP, respectively. However, unlike Simba, which employs homogeneous chiplets, the proposed MOHaM framework supports heterogeneous chiplets to ensure adaptability with diverse DNNs. Existing literature has many promising accelerators that could be used as SA chiplets in MOHaM for DSE [21][3][22][23][24][25][26].

2.2 Multi-DNN Workloads

To maintain competitive performance, emerging applications started employing DNNs for performing tasks like image classification, image segmentation, object detection, etc. For example, an AR/VR application could employ diverse DNNs together for different tasks, thus forming a multi-DNN workload. This diversity of DNNs leads to extreme heterogeneity in layer shapes and operations. Hence, different layers of DNNs could have specific dataflow preferences. However, most of the SAs are usually optimised for some specific layers with a fixed dataflow [8]. Figure 1a shows how the normalised EDP varies when the same DNN is run on different SAs. Here, three dominant layers, Convolution *CONV_0* and *CONV_33*, and Fully Connected *FC_121* from the ResNet50 DNN model [27] are considered. These layers are individually run on row-stationary

(Eyeriss [20]), weight-stationary (Simba [3]) and output-stationary (ShiDianNao [26]) SAs. While similar observations are also available in the literature [8][28][17][15], but the objective here is to advocate for flexible accelerators. MOHaM offers dataflow flexibility with heterogeneous SAs.

2.3 Hardware-Mapping Co-Optimisation

A typical hardware-mapping co-optimisation framework takes as input, a target DNN, an optimisation objective (e.g. latency) and the resource constraints (e.g. area). It returns as output a hardware configuration with the optimal instances of resources and an optimal mapping strategy to run the DNN on the accelerator. This co-optimisation can be used at *design-time*. Figure 1b shows how the normalised EDP varies when the same ResNet50 DNN model [27] is run with hardware-only, mapping-only and hardware-mapping co-optimisation. Hardware-only optimisation has lower energy due to optimal hardware resources but higher latency (delay) due to fixed mapping. On the contrary, mapping-only optimisation has a lower delay due to optimal mapping strategy but higher energy due to fixed hardware. Co-optimisation gets the best of hardware and mapping, and has the lowest normalised EDP. However, the search space is extremely large as it is the cross-product of hardware and mapping. For example, DiGamma [19] recently reported a design space as large as $O(10^{36})$, and that is only for a single accelerator. MOHaM offers a DSE which is orders of magnitude faster than the standard exhaustive search.

2.4 Multi-Objective Exploration

The convenient way of aggregating multiple optimisation objectives into one (mono-objective) invites penalty when objectives conflict. For example, due to the chosen memory hierarchy, if the system bandwidth is insufficient to feed enough data into the compute (MAC) units, they remain underutilised. In such a scenario, two mapping strategies may output identical latency but varying hardware resource utilisation, hence different energy and area. Similarly, it is possible to either access an enlarged on-chip buffer or the next-level buffer (memory) within the same energy budget. In such a scenario, even two hardware-mapping co-optimisations may output identical energy but different latency and area. These scenarios can be expressed as a set of non-dominating design points (multi-objective). Figure 1c shows the DSE when the same ResNet50 DNN model [27] is run for mono-objective exploration with EDP and multi-objective exploration with latency, energy and area. While mono-objective offers an aggregated solution, the Pareto set of distinct multi-objective solutions allows simultaneously exploring accelerator designs with varying requirements. MOHaM offers multi-objective exploration-enabled DSE.

3 Problem Formulation

Given a set of DNNs to be executed in parallel, and a library of heterogeneous accelerator templates, MOHaM aims to identify a Pareto set of system configurations with associated scheduling for the optimal trade-off between latency, energy and area[2]. Figure 2 shows MOHaM's internal organisation with inputs and output and are explained here.

2. Only for the sake of evaluation, latency, energy and area objectives are chosen in the problem formulation. The DNNs, accelerator templates and objectives can be modified in MOHaM for specific DSE.

3.1 MOHaM Inputs

The set of DNNs to be executed in parallel is called an Application Model, *AM*. While layers of different DNNs can be executed in any order, layers within the same DNN must be executed as per their dependencies. Figure 2(a) shows an *AM* with three DNNs, where arrows indicate the order of layer dependencies. For example, in *DNN0*, layer *L0.3* must be executed only after the completion of *L0.1* and *L0.2*.

The library of heterogeneous accelerator templates is a collection of parameterised, reconfigurable DNN accelerators used as chiplets. Each is referred to as a Sub-Accelerator Template, *SAT*, but when its parameters are set, it becomes a Sub-Accelerator Instance, *SAI*. Figure 2(b) shows a library of three *SAT*s. Eyeriss [20], Simba [3] and ShiDianNao [26] *SAT*s are used to evaluate MOHaM (refer Table 4). For example, in the Eyeriss *SAT*, if the number of PEs, shared buffer size and PE scratchpad size are set to something within their maximum bound, it becomes an Eyeriss *SAI*.

3.2 MOHaM Output

A Pareto set of solutions is identified with optimised objectives, latency, energy and area. Figure 2(c) shows a Pareto set, where each solution is a pair of a Multi-Accelerator System, *MAS*, and its associated Schedule, *S*. A *MAS* defines the hardware configuration, like the set of *SAI*s, Memory Interfaces, *MI*s, and their placement in the NoP. Figure 2(d) shows a *MAS* with four heterogenous *SAI*s and an external memory (e.g., DRAM) interconnected by a 2D Mesh NoP.

A schedule *S* defines the mapping strategy, like where each layer of every DNN is mapped in the chiplets, and the order in which they will be executed. Figure 2(e) shows a schedule, where multiple layers are mapped onto the same chiplet (or *SAI*). For example, *L2.0* and *L1.1* are mapped onto *SAI0.0*. In such a case, they are executed sequentially. Each layer gets the mapping information in a loop-nest form.

3.3 MOHaM Working

MOHaM takes each layer of the *AM* and maps them to each of the *SAT*s from the input library. It identifies the Pareto set of mappings for each layer across all the *SAT*s with respect to the chosen optimisation objectives. MOHaM then employs a custom GA to conduct multi-objective optimisation with respect to the distinct objectives, latency, energy, and area. This optimisation fine-tunes the (a) selection of Pareto mappings for each layer identified initially, (b) layer scheduling strategies, (c) instantiations of *SAT*s, (d) allocation of hardware resources to each *SAI*, (e) assignment of layers to *SAI*s, and (f) placement of *SAI* tiles in the NoP.

3.4 MOHaM Problem Formulation

Based on the above information, the problem can be formulated as follows: MOHaM aims to minimise latency, energy, and area while ensuring they remain distinct from one another, thereby adopting a multi-objective optimisation.

$$\min Latency(AM, MAS, S)$$
$$\min Energy(AM, MAS, S)$$
$$\min Area(MAS)$$

While latency and energy are influenced by factors such as the application model, the configuration of the multi-accelerator system, and the schedule, area is solely determined by the configuration of the multi-accelerator system.

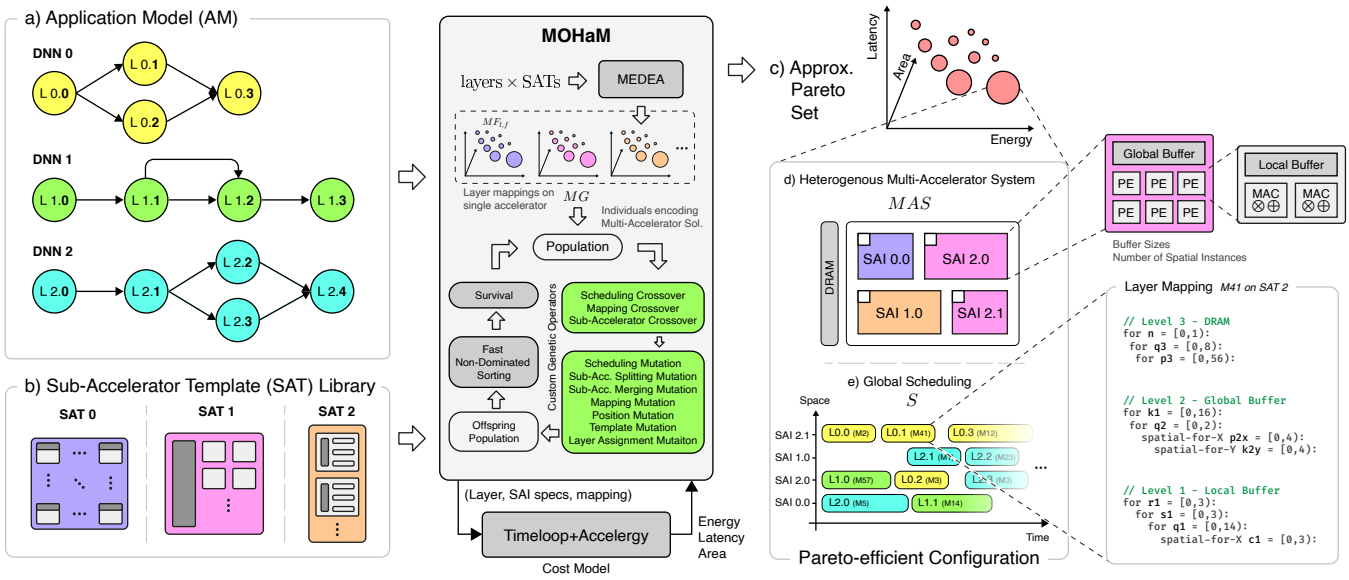


Figure 2: Inputs, output and internal organisation of the proposed MOHaM framework.

Schedule S provides a set of pairs *(L, SAI)*, indicating which layer is mapped to which SA instance along with the order of their execution. A layer can not be mapped into more than one SA instance. It can be represented by:

$$\forall\, t \nexists\, (L_i, SAI_i) \in S \wedge (L_j, SAI_j) \in S$$

such that

$$L_i = L_j \wedge SAI_i \neq SAI_j$$

3.5 MOHaM Search Space

Let *Num. Layers* and *Num. DNNs* be the number of layers and DNNs, respectively, that constitute an application model. Let *Max. Chiplets* be the maximum chiplets that can be used in a multi-accelerator system. Let *Num. SATs* be the number of templates available in the *SAT* library. Let *Num. Params* and *Num. Values* be the number of free parameters and the number of values each parameter can take, respectively, of a *SAT*. For the sake of simplification, it is assumed that *Num. Params* and *Num. Values* are the same for all the *SATs*[3]. A lower bound for the search space size is:

$$Search\ Space\ Size \geq \left(7! \times \prod_{d=1}^{7} m^{|F_d|}\right)^{Num.\ Layers} \quad (1)$$

$$\times \sum_{Chiplets=1}^{Max.\ Chiplets} \left(Num.\ SATs^{Chiplets}\right. \quad (2)$$

$$\times\ Num.\ Values^{Num.\ Params\ \times\ Chiplets} \quad (3)$$

$$\times\ Chiplets! \quad (4)$$

$$\left.\times\ Chiplets^{Num.\ Layers}\right) \quad (5)$$

$$\times\ Num.\ DNNs! \quad (6)$$

Assuming fixed-dataflow *SAT*s, term (1) is the mapping space size calculated by multiplying the number of loop permutations in DRAM (assuming these are not fixed) by the number of possible factorisation of each workload dimension. To calculate the latter, loop tiling can be thought of as the task of assigning a tiling level to each prime factor of the workload dimensions involved. Without spatial unrolling, the number of tiling levels coincides with the number of memory levels; otherwise, it will be greater. In term (1), 7 is the number of tensor dimensions in a convolutional layer, m is the number of tiling levels (greater or equal to the number of levels in the *SAT* memory hierarchy), and F_d is the set of the prime factors of the d^{th} workload dimension.

For a multi-accelerator system consisting of *Chiplets* chiplets, term (2) represents the number of possible combinations in which *SAT*s can be assigned to the chiplets.

Term (3) represents the number of possible combinations of the various instances of *SAT*s that can be assigned to the chiplets. As mentioned above, here it is assumed that *Num. Params* and *Num. Values* are uniform across all the *SAT*s. However, in the generic case, each parameter p will vary within a specific set with a cardinality of *Num. Values(p)*. Hence, for the generic case, term (3) will be modified as:

$$\left(\prod_{p=1}^{Num.\ Params} Num.\ Values(p)\right)^{Chiplets} \quad (7)$$

Term (4) represents the possible combinations of assigning a chiplet to an NoP tile. Term (5) represents the number of possible combinations for assigning a layer to a chiplet. Finally, term (6) gives a lower bound on the possible schedules considering the combinations of execution order at the DNN granularity rather than the layer granularity.

The simulation configuration used to evaluate MOHaM in Section 5 has *Num. Layers* as 200 (average), *Num. DNNs* as 4, *Max. Chiplets* as 8, *Num. SATs* as 3, *Num. Params* as 4 (average), and *Num. Values* as 4096 (average). Hence, the search space size is in the order of 10^{4318}. To get an idea of how large this is, consider that the number of particles in the observable universe is estimated to be around 10^{80}.

4 The MOHaM Framework

To search such an enormous space, the proposed MOHaM framework adopts a two-step approach, *layer mapping* and *global scheduling*. Algorithm 1 presents a high-level flow of

3. It compacts term (3) and avoids its expansion for each *SAT*.

Algorithm 1: MOHaM high-level flow

```
1  Procedure LayerMapper(AM, SSAT)
2      MG← {}
3      for layer in UniqueLayers(AM) do
4          ML← {}
5          for arch in SSAT do
6              MF← RunMEDEA(layer, arch)
7              ML.Add(MF)
8          MG.Add(ML)
9      return MG
10 Procedure GlobalScheduler(AM, SSAT, MG)
11     PP← InitialPopulation()
12     for g ← 1 to G do
13         // G = number of generations
14         OP← ApplyCrossoverOperators(PP)
15         OP.ApplyMutationOperators(OP)
16         OP.Evaluate()
17         MP← FastNonDominatedSort(PP, OP)
18         OP← Survival(MP)
19     return PP.GetParetoIndividuals()
20 Procedure MOHaM(AM, SSAT)
21     MG← LayerMapper(AM, SSAT)
22     ST← GlobalScheduler(AM, SSAT, MG)
23     return ST // Pareto set of schedules
```

Table 2: Acronyms, abbreviations and key terms

AM	Application Model; a set of independent DNNs.
SAT	Sub-Accelerator Template; a parameterised and reconfigurable DNN accelerator with a fixed memory hierarchy.
SAI	Sub-Accelerator Instance of an SAT with fixed parameters.
MI	Memory Interface; connects memory banks with SAIs.
NoP	Network-on-Package; interconnect SAIs and MIs.
MAS	Multi-Accelerator System; a set of SAIs with memory.
S	Schedule; spatial and temporal allocation of each layer of every DNN of an AM on the SAIs of a MAS.
$\mathbf{M}_{l,f,i}$	i^{th} Pareto mapping for a layer *l* in an SAT *f*.
$\mathbf{MF}_{l,f}$	Pareto set of mappings for a layer *l* in an SAT *f*.
$\mathbf{ML}_{l}$	Pareto set of mappings for a layer *l* in all the SATs.
MG	Pareto set of mappings for all the layers in all the SATs.
NSGA-II	Non-Dominated Sorting Genetic Algorithm [16].
Gene	A tuple of encoded values for a layer or an SAI.
Genome	An array of genes that represent all the information about a layer or an SAI, like mapping, execution order, etc.
Chromosome	A concatenation of genomes of all the layers and SAIs.
Individual	A chromosome with valid mapping, SAI and execution order of the layers, and SAT and NoP tiles for the SAIs.
Population	A set of individuals evolving through GA generations.
Pareto Efficiency	An individual is Pareto-efficient if no one else in the population is better for all the objectives at the same time.
Topological Sort	A linear ordering of a Directed Acyclic Graph (DAG) where a node appears before all the nodes it points to.

MOHaM, and the following sections describe its working. Table 2 presents the acronyms, abbreviations and key terms.

4.1 Layer Mapper

In this step, each layer of the *AM* is mapped onto each of the *SAT*s from the input library. At all times, a layer is mapped onto any single instance of an *SAT* and can not be split across multiple *SAT* instances. Two layers, l_i and l_j can be instances of the same workload, i.e., have the same problem dimensions. Hence, to reduce the layer mapping search space, MOHaM only maps the unique layers to *SAT*s.

Let $M_{l,f,i}$ be the i^{th} Pareto mapping for a layer l of the *AM* in a *SAT* f from the input library. $M_{l,f,i}$ is a choice of a specific tiling, loop ordering, parallelism and clustering in f. Let $MF_{l,f}$ be the Pareto set of mappings with respect to the objectives for l in f. $MF_{l,f}$ can be represented by:

$$MF_{l,f} = \{\, M_{l,f,i} \mid i = 0, \ldots, (m_{l,f} - 1) \,\} \tag{8}$$

where, $m_{l,f}$ is the total number of Pareto mappings. Now, let ML_l be the Pareto set of mappings for l in all the *SAT*s of the input library. ML_l can be represented by:

$$ML_l = \{\, MF_{l,f} \mid f = 0, \ldots, (F - 1) \,\} \tag{9}$$

where, F is the total number of *SAT*s in the input library. Finally, let MG be the Pareto set of mappings for all the layers of *AM* in all the *SAT*s. MG can be represented by:

$$MG = \{\, ML_l \mid l = 0, \ldots, (L - 1) \,\} \tag{10}$$

where, L is the total number of layers in the *AM*. MOHaM obtains the Pareto set of mappings in equation (8) to (10) by leveraging MEDEA [29] (refer line 6 in Algorithm 1). For each layer of the *AM*, MEDEA searches for the Pareto set of mappings onto an *SAT* instance. The Pareto set is for the chosen optimisation objectives, latency, energy and area.

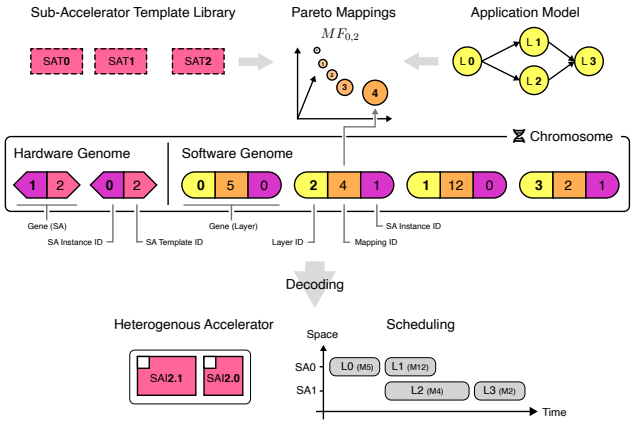


Figure 3: MOHaM global scheduler chromosome structure.

4.2 Global Scheduler

In this step, the Pareto set of mappings found for each layer in the previous step (Section 4.1) is used to search the global scheduling. It returns a Pareto set of (*MAS*, *S*) pair with minimum latency, energy and area. This scheduler exploits one of the most widely accepted multi-objective GA, NGSA-II [16]. It has five major phases, *sampling*, *selection*, *crossover*, *mutation* and *survival*. While the original selection and survival phases are used, several custom genetic operators are designed for problem-specific crossover and mutation to increase sampling efficiency and decrease search time.

4.2.1 Chromosome Encoding

A fundamental step in employing a GA is to define an encoding for the individuals in the population. MOHaM requires this encoding to represent, *(a)* mapping strategy of each layer, *(b) SAI* of each layer, *(c)* execution sequence of layers in the *AM*, *(d)* SA template of each *SAI*, and *(e)* NoP tile of each *SAI*. A two-part chromosome used by the scheduler is shown in Figure 3 and is described here:

- **Hardware Genome:** It encodes the instances of the *SAT*s. It is an array of genes, where each gene denotes an SAI. Each gene is a tuple $\langle SAI, SAT \rangle$ where *SAI*

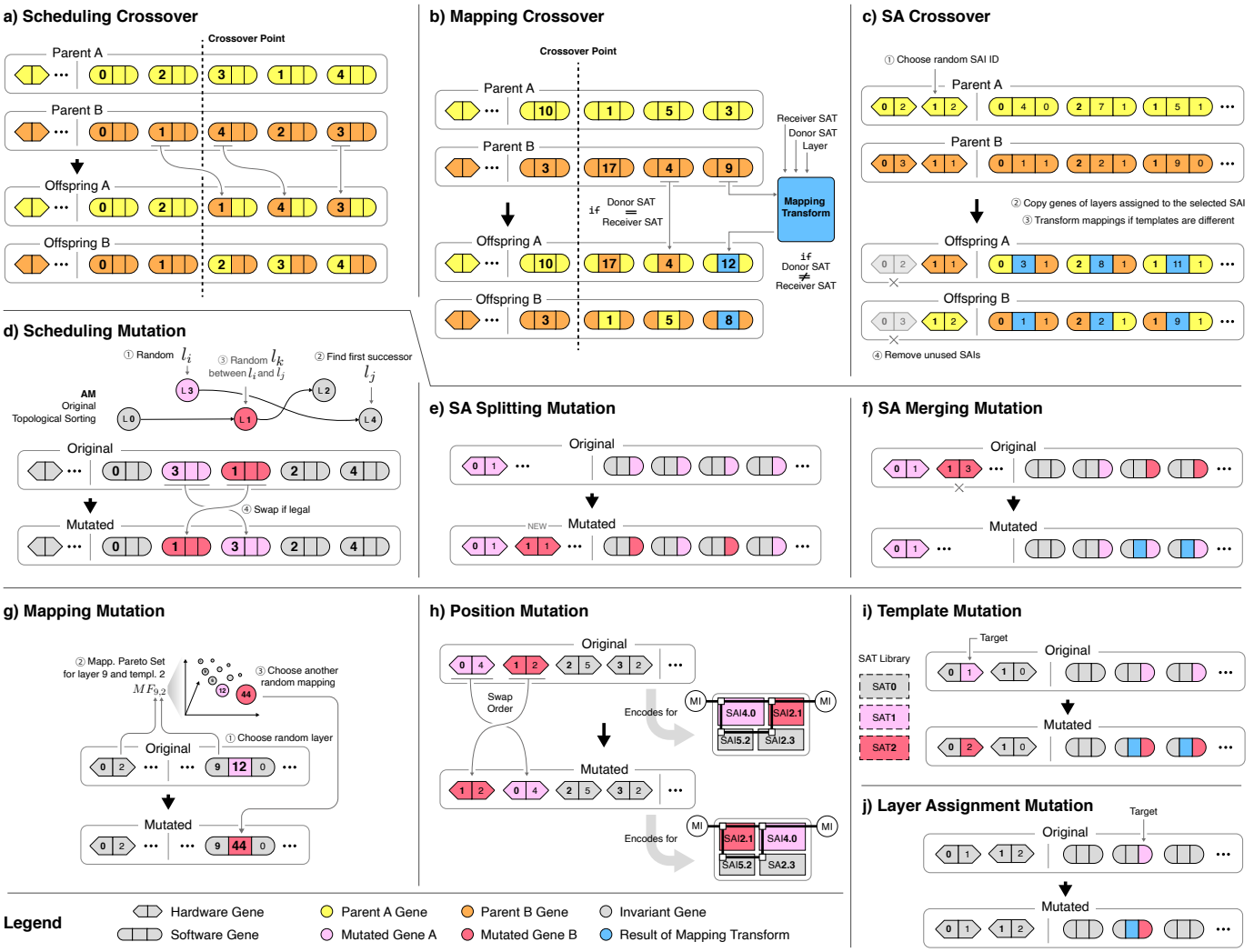


Figure 4: MOHaM-specific genetic operators

is the SA instance and *SAT* is the template identifier. The order of the genes represents the position of the *SAI* hosting tile in the NoP. The number of genes is equal to the number of *SAI*s and varies between 1 and maximum NoP tiles across chromosomes.

- **Software Genome:** It encodes the layers of the *AM*. It is an array of genes, where each gene denotes a layer. Each gene is a tuple $\langle LID, MID, SAI \rangle$ where *LID* is the layer identifier, *MID* is the mapping identifier, and *SAI* is the SA instance the layer will be executed. The order of the genes is a topological sorting of the layers using Kahn's algorithm [30] and represents the temporal sequence in which the layers will be executed. The number of genes is equal to the number of layers in the *AM* and is fixed for chromosomes.

4.2.2 Custom Genetic Operators

The standard crossover swaps parental genes at a pivot point to generate offspring, while standard mutation randomly changes some genes of the offspring to introduce variation. However, directly applying these operators to MOHaM will not work since offspring chromosomes must adhere to specific constraints to be valid individuals. Each MOHaM chromosome is a tuple $\langle genome^{hw}, genome^{sw} \rangle$ where $genome^{hw}$ can be represented by:

$$genome^{hw} = \{\, g_i^{hw} = \langle SAI, SAT \rangle \mid i = 1, \ldots, Chiplets \,\}$$

Whereas $genome^{sw}$ can be represented by:

$$genome^{sw} = \{\, g_j^{sw} = \langle LID, MID, SAI \rangle \mid j = 1, \ldots, Layers \,\}$$

The constraints for valid individuals are specified as follows:

C1 In the hardware genome, the *SAT* of a gene must refer to one of the templates from the input library.

$$g_i^{hw} = \langle \bullet, SAT_i \rangle \mid SAT_i \in SAT\ Input\ Library$$

C2 In the hardware genome, each *SAI* must be referred to by at least one gene from the software genome.

$$g_i^{hw} = \langle SAI_i, \bullet \rangle \mid \exists j\ with\ g_j^{sw} = \langle \bullet, \bullet, SAI_i \rangle$$

C3 In the software genome, the *MID* of a layer l must refer to one of the mappings from $MF_{l,f}$, where f is the template of the *SAI* on which l will be executed.

$$g_j^{sw} = \langle \bullet, MID_j, SAI_j \rangle \mid MID_j \in MF_{l_j, f_j} \quad where\ f_j = SA\ Template(SAI_j)$$

C4 In the software genome, the *SAI* of a layer must refer to one of the encoded genes in the hardware genome.

$$g_j^{sw} = \langle \bullet, \bullet, SAI_j \rangle \mid \exists i \; with \; g_i^{hw} = \langle SAI_j, \bullet \rangle$$

C5 In the software genome, the order of the genes must be a valid topological sorting that respects the dependencies between layers of the DNNs from the AM.

$$\forall j, k = 1, \ldots, Layers \mid j < k \\ g_j^{sw} = \langle LID_j, \bullet, \bullet \rangle, \; g_k^{sw} = \langle LID_k, \bullet, \bullet \rangle \\ \Rightarrow (LID_k, LID_j) \notin D$$

where D is the set of dependencies between layers.

Therefore, the choice of crossover and mutation operators significantly impacts the performance of a GA, and it is often necessary to adapt them to the specific problem at hand. While there is no systematic way, using domain-specific knowledge of the problem is a common practice for customising crossover and mutation. Hence, a set of custom crossover and mutation operators are designed by considering the problem-specific constraints to (a) explore the search space strategically, and (b) increase the sampling efficiency. They are shown in Figure 4 and described here:

(a) Scheduling Crossover: It combines the topological sorting of the parent chromosomes, as shown in Figure 4(a). It generates offspring by taking the first part of one of the parents, i.e., all the genes before the crossover point, and appending all the unique genes from the other parent. Existing literature [31] has proof that by proceeding this way, the offspring will also have valid topological sorting.

(b) Mapping Crossover: It combines the mappings of the parent chromosomes, as shown in Figure 4(b). It generates offspring by taking layer mappings from the first part of one of the parents, i.e., all the genes before the crossover point, and the remaining mappings from the other parent. However, a new mapping might be invalid if it is for an SA template different from that of the *SAI* on which the layer will be executed. In such a case, a compensation mechanism called *Mapping Transform* finds the most similar one among all the possible mappings for the layer in that *SAI*.

(c) SA Crossover: It swaps a random *SAI* at i^{th} position, sai_i, between the parent chromosomes, as shown in Figure 4(c). If both parents *A* and *B* have an *SAI* at the i^{th} position, they are swapped and two offspring are generated. In such a case, if sai_i of *A* and sai_i of *B* are of different SA templates, all the mappings from both of their layers undergo *Mapping Transform*. Whereas, if only one parent has an *SAI* at the i^{th} position, it is added to the other parent with all its assigned layers and only one offspring is generated.

(d) Scheduling Mutation: It mutates the topological sorting of a chromosome, as shown in Figure 4(d). Let l_i be a random gene (layer) from the software genome. Let l_j be the nearest layer in the traversal that is dependent on l_i. Let l_k be a random layer between l_i and l_j. If all the layers on which l_k has a dependency lie before l_i in the traversal, their position can be swapped. Similar to 4(a), the mutated chromosome will also have a valid topological sorting [31].

(e) SA Splitting Mutation: It reduces the load in a random *SAI* at i^{th} position, sai_i, to mutate a chromosome, as shown in Figure 4(e). A new sai_j of the same SA template is appended to the hardware genome. Thereafter, half of the layers currently assigned to sai_i are randomly chosen and assigned to sai_j. The goal is to increase parallelisation.

(f) SA Merging Mutation: It increases the load in a random *SAI* at i^{th} position, sai_i, to mutate a chromosome, as shown in Figure 4(f). Another existing *SAI* at j^{th} position, sai_j, is randomly chosen and all its layers are assigned to sai_i. If sai_j is of a different SA template than sai_i, all the mappings of the imported layers undergo *Mapping Transform*. The goal is to reduce chip area with reduced *SAI*s.

(g) Mapping Mutation: It modifies the *MID* of a random layer l to mutate a chromosome, as shown in Figure 4(g). All the possible mappings of l are in the $MF_{l,f}$ Pareto set, where f is the template of the *SAI* on which l will be executed. One of those possible mappings is assigned to l for mutation.

(h) SA Position Mutation: It swaps the position of two *SAI* hosting tiles in the 2D Mesh NoP, as shown in Figure 4(h). The goal is to find a configuration where the system bandwidth is distributed among the *SAI*s and *MI*s in a way that avoids communication bottlenecks and reduces latency.

(i) SA Template Mutation: It modifies the SA template of a random *SAI* to mutate a chromosome, as shown in Figure 4(i). All the mappings from different layers to be executed on the mutated *SAI* undergo *Mapping Transform*.

(j) Layer Assignment Mutation: It modifies the *SAI* of a random layer l to mutate a chromosome, as shown in Figure 4(j). If the modified *SAI* is of a different SA template, the corresponding mapping for l undergoes *Mapping Transform*.

These operators are input agnostic, i.e., MOHaM can be used for any DNN as an *AM*, with any accelerator as an *SAT*, without modifying or adding any custom GA operator.

4.3 Objectives Evaluation

Three objective metrics are evaluated for each individual: (a) *latency*, which is the makespan of the workload, (b) *energy*, which is the total energy consumption to execute the workload, and (c) *area*, which is the real estate of the hardware for the workload. They depend on the system configuration and the schedule. For example, increasing the number of *SAI*s decreases latency but increases area. Similarly, the mapping strategy for each layer of the *AM*, and the *SAT*s affect all three objectives. MOHaM translates the chromosome encoding into objective metrics using the Timeloop [17] + Accelergy [18] cost model (refer Figure 2).

The translation begins from the hardware genome. Each of its genes is converted into an *SAI* of the appropriate *SAT*. At this point, parameter values of the *SAI*s are unknown. Then software genomes are examined to identify which mappings are used for each gene (layer). All the free parameters, including the number of PEs, size of buffers, etc., are set to the maximum required by the mappings for all the layers assigned to an *SAI*. For example, Figure 2(e) shows that $L2.0$ and $L1.1$ are assigned to *SAI0.0*. If they require 64 and 128 PEs and 64 KiB and 32 KiB shared buffers, respectively, *SAI0.0* will be assigned 128 PEs and 64 KiB shared buffer. Each *SAI* executes its assigned layers, and the total area is estimated. This over-provision is automatically handled, as MOHaM finally identifies individuals with minimum area.

As each mapping has its corresponding energy consumption, summing them for all the layers could provide the total energy consumption. However, energy for reads

and writes in the buffers depends on their sizes. Hence, energy for each mapping must be estimated based on the updated values of the free parameters in the *SAI*s.

Latency is estimated using the topological sorting encoded in the software genome. Each of its genes (layers) is read sequentially and mapped on the assigned *SAI*s. This traversal guarantees that a layer's dependencies are already scheduled before its turn comes. Each layer has a start and end time, and the total latency is estimated based on the latest end time among all the layers. However, this is true only when no communication or memory bottlenecks exist.

4.3.1 Communication Modeling

As shown in Figure 2(d), MOHaM assumes that a 2D Mesh NoP interconnects the *SAI*s and memory banks. The *SAI*s communicate with the memory banks through the *MI*s. The NoP is implemented via on-package links in a passive silicon interposer. It employs efficient intra-package signalling circuits using Ground-Reference Signaling (GRS) technology. Specifically, each chiplet has eight chiplet-to-chiplet GRS transceivers; four transmitters and four receivers. A transceiver has four data lanes, each providing 4 GB/s, thus a total peak chiplet bandwidth of 4 * 4 GB/s = 16 GB/s and energy of 0.82 pJ/bit [3]. The NoP employs the XY routing algorithm. There is no inter-chiplet communication and the *SAI*s only communicate with the *MI*s. While the positions of the *MI*s are configurable, MOHaM assumed one at each corner of the 2D Mesh NoP for the evaluation in Section 5. *SAI*s reads and writes in their nearest *MI*s. Some CPU processing happens between layer execution (e.g. tensor reordering), and the output is stored in the nearest memory bank of the *SAI* executing the next layer. Depending on their position in the NoP, multiple *SAI*s could be assigned the same *MI*, thus competing for the shared link. Let 16 *SAI*s be arranged in a 4×4 2D Mesh NoP with 4 *MI*s, one at each corner. Here, 4 *SAI*s will be sharing 1 *MI* and the NoP link connecting that *MI* will be the bottleneck link. So, either the shared *MI* bandwidth (BW_{MI}) or the bottleneck link bandwidth (BW_{BL}) will be the reason for the congestion.

Each layer has a certain read and write bandwidth requirement when executed on its assigned *SAI*. If a time segment has parallel execution of layers and the combined bandwidth requirement of the layers executed on the *SAI*s sharing the same *MI* exceeds either the BW_{MI} or the BW_{BL}, there will be stalls. Let the time segment be n cycles and the combined bandwidth requirement be BW_{req}. The stalled time segment of m cycles can be calculated by:

$$m = n \times \frac{BW_{req}}{min(BW_{MI}, BW_{BL})}$$

$m > n$ signifies a delay in the execution of certain layers. In such cases, MOHaM compensates the start times of all the subsequent layers with dependencies on the delayed layers. However, if a subsequent layer is executed on an *SAI* that communicates with a different *MI*, the delay in its start time might create a new stalled time segment. MOHaM considers all such cases to correctly estimate the latency. After evaluating all the stalled time segments, the latest end time among all the layers becomes the new total latency.

To estimate the communication energy, MOHaM considers the amount of data transferred between an *SAI* and its nearest *MI* and the hop distance between them following the XY routing algorithm. It can be calculated by:

$$\sum_{i=1}^{Num.\ SAIs} \Big(Bits.Transferred_{SAI_i} \times Hops_{XY}(SAI_i, MI) \times Energy/Hop/Bit \Big)$$

The NoP real estate is not currently considered in the area estimation. MOHaM considers a silicon interposer-based NoP which is outside the chiplet areas. Hence, it is ignored while estimating the area. Nevertheless, it can be easily included in the estimation by multiplying the number of *SAI*s with a NoP router area. Depending on the number of ports, different router areas can also be considered.

4.3.2 Convergence Criterion

The proposed MOHaM framework supports a GA stopping criterion based on the density of the non-dominated solutions presented in [32]. Alternatively, simulating for a fixed number of generations can also be configured very easily.

5 Evaluation

This section presents multiple experiments conducted to evaluate the performance of the MOHaM framework.

5.1 Methodology

5.1.1 DNNs and Workloads

MOHaM is to be used at design-time of a multi-accelerator system, where the application domains will be known or guessed apriori. Hence, AR/VR, edge, and mobile workloads are considered for evaluation. Besides, standard data center workloads are dominated by vision, language and recommendation-based DNNs [33]. Hence, MOHaM is also evaluated with such a workload to show its effectiveness if the application domains can be guessed. All of them are multi-DNN workloads created with models from the MLPerf benchmark suite [34][35]. Table 3 presents the workload scenarios along with their DNN models. MOHaM takes them as inputs in the ONNX [36] interoperable format.

Table 3: Multi-DNN workload scenarios

#	Workload	Domain	DNN Model
A	AR/VR	Image Classification Image Segmentation Object Detection	ResNet50 UNet SSD-MobileNetV1 YOLOv3
B	Edge	Image Classification Object Detection Language Processing	ResNet50 SSD-ResNet34 BERT-Large
C	Mobile	Image Classification Image Segmentation Language Processing	MobileNetV3L DeepLabV3+ MN2 Mobile-BERT
D	Data Center	Image Classification Object Detection Language Processing Recommendation	GoogleNet YOLOv3 BERT-Large DLRM

5.1.2 Sub-Accelerator Templates

The following state-of-the-art constitutes the *SAT* library[4]:

- Eyeriss [20]: Row-stationary dataflow
- Simba [3]: Weight-stationary dataflow

4. Any accelerator can be added in the template library of MOHaM.

Table 4: MOHaM simulation configuration

Exploration Parameters			
Num. Generations	300	Population Size	250
Sched. Cross. Prob.	0.103	Sched. Mut. Prob.	0.052
SA Cross Prob.	0.045	Template Mut. Prob.	0.041
Merging Mut. Prob.	0.042	Splitting Mut. Prob.	0.039
Mapping Mut. Prob.	0.048	Mapping Cross. Prob.	0.047
Layer Assign. Mut. Prob.	0.025	Position Mut. Prob.	0.027
Max. SA Instances	8		
Common Architecture Parameters			
Technology Node	45 nm	DRAM Technology	LPDDR4
Mem. Interface BW	4 GB/s	Clock Frequency	1 GHz
Word Size	8 bits	SRAM Buf. BW	16 GB/s
Eyeriss-like Template			
Dataflow	Row-Stat.	Max. Num. of PEs	4096
Max. Shared Buf. Size	131 KiB	Max. PE Scratchpad Size	0.5 KiB
Simba-like Template			
Dataflow	Weight-Stat.	Max. Num. of PEs	64
Max. MACs per PE	64	Max. Global Buf. Size	64 KiB
Max. Weight Buf. Size	32 KiB	Max. Input Buf. Size	8 KiB
Max. Accum. Buf. Size	3 KiB		
ShiDianNao-like Template			
Dataflow	Output-Stat.	Max. Num. of PEs	4096
Max. Neurons Buf. Size	131 KiB	Max. Synapses Buf. Size	131 KiB

- ShiDianNao [26]: Output-stationary dataflow

They are heterogeneous accelerators and are chosen to support diverse DNNs. Table 4 presents MOHaM configuration for the experiments, where the GA exploration parameters are based on the guidelines of the widely adopted [37], and the architectural parameters are based on the state-of-the-art [20][3][26]. For the parameter names beginning with "*Max.*", the presented values are their upper bound and are reconfigurable. For a chosen Pareto solution, two *SAI*s of the same *SAT*, say Eyeriss, can have different values for the number of PEs, shared buffer size and PE scratchpad size. The upper bound is to limit the search space for exploration.

5.1.3 Solution Anatomy

Figure 5 shows the visual solution anatomy available in MOHaM. It shows the scheduling Gantt chart for latency, and two pie charts for energy and area of a Pareto solution from the AR/VR workload. The Gantt chart shows, on the y-axis, the *SAI*s, and on the x-axis, the start and end times of the execution of each layer, measured in cycles. Each bar represents the execution of a layer of the *AM* on a particular *SAI*. Black traces represent bandwidth-constrained execution segments as described in Section 4.3.1. The latest end time among all the layers is the latency (makespan) of the workload. The area pie chart displays each *SAI*'s contribution to the real estate of the *MAS*. Instances from the same *SAT* have the same colour. The energy pie chart displays each DNN's contribution to the total energy consumption to execute the workload. The Pareto set might have another solution with better scheduling and silicon utilisation. The solution anatomy helps in comparing design points.

5.2 Results

For all the results shown in Figures 7, 6, 8 and 9, the x-axis represents energy in *mJ*, the y-axis represents latency in *cycles*, and the size of the design points represents the area in mm^2. A common observation valid for all these results is the distribution of the solutions found by MOHaM. They are spread over a large Pareto surface rather than limited to a specific region. This is an important advantage of MOHaM as it provides a variety of trade-off solutions to choose from.

5.2.1 Homogeneous vs Heterogeneous Accelerators

This experiment evaluates the need for heterogeneous sub-accelerators (SAs) in multi-accelerator systems. Figure 6 shows the comparison of Pareto solutions with homogeneous and heterogeneous SAs. For homogeneous SAs, MOHaM is run once, each with only Eyeriss-like, Simba-like, and ShiDianNao-like SA templates. Then, they are compared with the result of a complete MOHaM run for heterogeneous SAs. It is observed that for the *Edge* and *Data Center* workloads, Eyeriss (yellow) has lower latency at the cost of higher energy and bigger area. On the contrary, ShiDianNao (red) is area efficient but at the cost of very high energy and latency. For the *AR/VR* and *Mobile* workloads, Simba (purple) has the best energy and latency, while ShiDianNao (red) has the worst latency, respectively. Among the solutions with homogeneous SAs, Simba (purple) generally has better energy, while ShiDianNao (red) has a better area. With heterogeneous SAs, the solutions (black) are more uniformly distributed. For the *AR/VR* workload, they have solutions with the lowest latency, energy, and area. For example, design point *A* from heterogeneous (black) can achieve a 89.4% latency reduction compared to design point *B* from Eyeriss (yellow). Heterogeneous SA-based solutions are equally good for other workloads. For example, for the *Data Center* and *Edge* workloads, design point *A* from heterogeneous (black) can achieve a 9.72% and 20.41% latency reduction, respectively, compared to design point *B* from Simba (purple). Besides, the lowest energy solutions by heterogeneous SAs can achieve a 24.7% reduction compared to Simba. This is possible as layers with specific dataflow preferences can be executed on the appropriate SAs. MOHaM instantiates the right set of SAs from the available templates and maps the layers for efficient execution. *The key takeaway from this experiment is that flexible dataflow support with heterogeneous SAs increases the scalability of a multi-accelerator system toward diverse and emerging workloads.*

5.2.2 Independent vs Simultaneous Optimisation

This experiment evaluates the need for hardware-mapping co-optimisation in multi-accelerator systems. Figure 7 shows the comparison of Pareto solutions with hardware-only, mapping-only and hardware-mapping co-optimisation. For hardware-only optimisation, MOHaM is run with only Simba-like SA templates to have a fixed dataflow (e.g., weight-stationary), similar to ConfuciuX [14]. For mapping-only optimisation, MOHaM is run with a fixed hardware configuration of eight heterogeneous SAs, similar to MAGMA [10]. Finally, they are compared with the result of a complete MOHaM run for hardware-mapping co-optimisation. It is observed that for the *AR/VR* workload, hardware-only optimisation (red) has a smaller area but high latency and energy. This is due to the fixed mapping (dataflow) strategy for all the layers. Whereas mapping-only optimisation (purple) has lower latency and energy but costs a bigger area. This is due to the fixed hardware configuration. In general, hardware-only optimisation has a smaller area but poor latency and energy across all workloads. Similarly, mapping-only optimisation has lower latency but a big area. With hardware-mapping co-optimisation, the solutions (black) are more uniformly distributed and achieve better

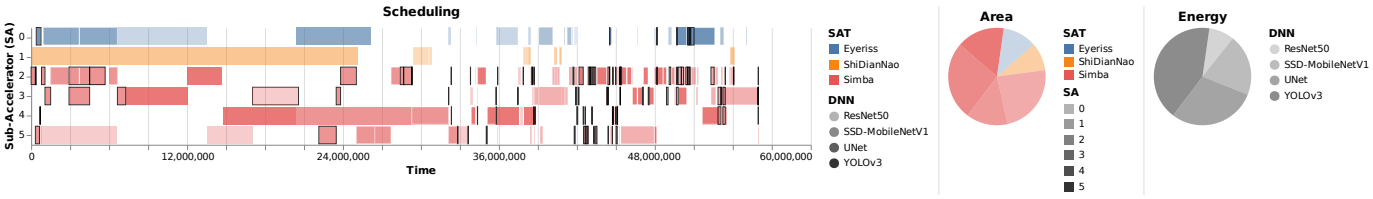


Figure 5: Scheduling Gantt chart for latency, pie charts for area and energy breakdowns of a Pareto solution.

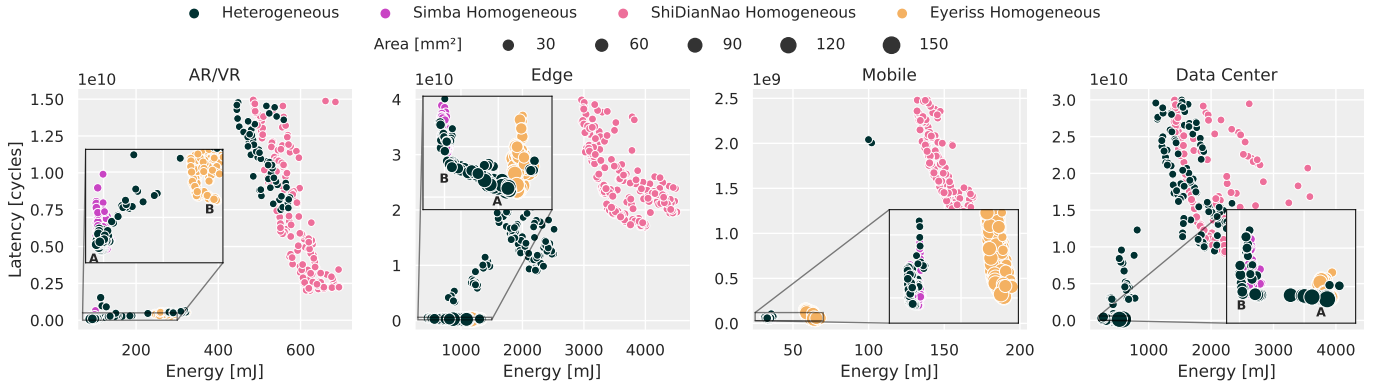


Figure 6: Comparison of Pareto solutions with homogeneous and heterogeneous sub-accelerators.

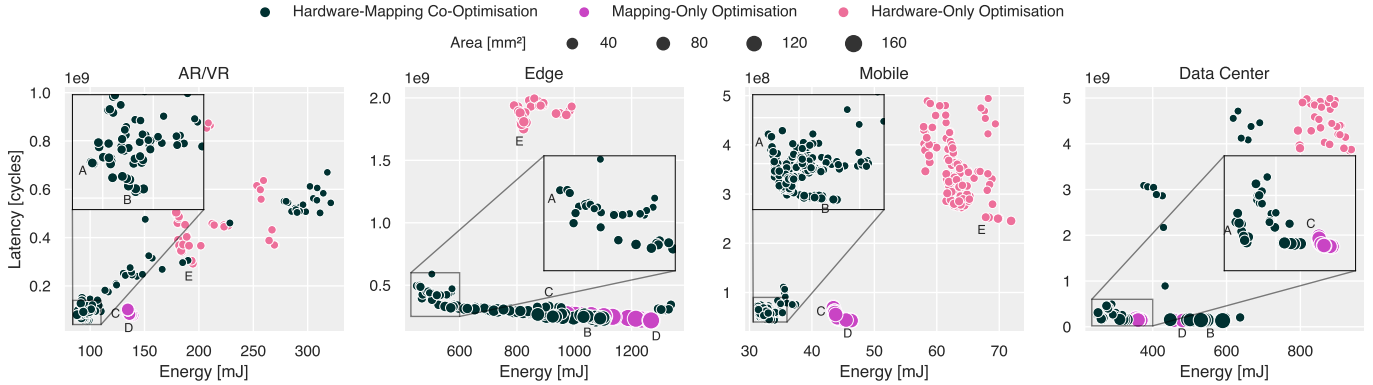


Figure 7: Comparison of Pareto solutions with hardware-only, mapping-only, and hardware-mapping co-optimisation.

performance. For example, for the *AR/VR* workload, design point A from co-optimisation (black) has two times the area as design point E from hardware-only optimisation (red) but has 55% less energy and 72% less latency. It is also observed that mapping-only optimisation achieves similar latency but with higher energy and a bigger area. For example, for the *Edge* workload, design point B from co-optimisation (black) has the same latency as design point D from mapping-only optimisation (purple) but has 15.3% less energy and 36.5% less area. Similar solutions exist in *Mobile* and *Data Center* workloads. This results from MOHaM instantiating the SAs with optimal hardware resources and mapping the layers per their dataflow preferences for optimal execution. *The key takeaway from this experiment is that hardware-mapping co-optimisation can support diverse workloads with the best overall performance in a multi-accelerator system.*

5.2.3 Single-Objective vs Multi-Objective Exploration

This experiment evaluates the need for multi-objective exploration in multi-accelerator systems. Figure 8 shows the comparison of individual solutions with mono-objective and Pareto solutions with multi-objective exploration. For mono-objective exploration, MOHaM is run once, each with only latency, energy, and EDP as an objective. Three Linear Combinations (LCs) of energy and latency with coefficients: $(0.25, 0.75)$, $(0.5, 0.5)$, and $(0.75, 0.25)$ are also run once. Then, they are compared with the result of a complete MOHaM run for multi-objective exploration. When objectives conflict, the best solution for one may sometimes lead to the worst for others with mono-objective exploration. For example, for the *Data Center* workload, the solution with energy as an objective (red) is at the extreme left (i.e., best), but the latency is relatively high. Similarly, the solution with latency as an objective (purple) is best, while the area is worst. Even a slightly better solution for one objective may sometimes lead to heavily penalising others with mono-objective exploration. For example, for the *Edge* workload, the design point with latency as an objective (purple) is only 3% better than design point A from multi-objective (black), but comes at the cost of 14.2% more energy and 30.1% more area. Furthermore, the best solution for one objective may sometimes be sub-optimal with mono-objective exploration. For example, for the *Mobile* workload, the design point with latency as an objective (purple) has 7.5% more latency, 34% more energy and 75% more area than design point A from multi-objective (black). Even the best solution for popular aggregated objectives, EDP, may sometimes be sub-optimal with mono-objective exploration. For example, compared to the design point with EDP as an

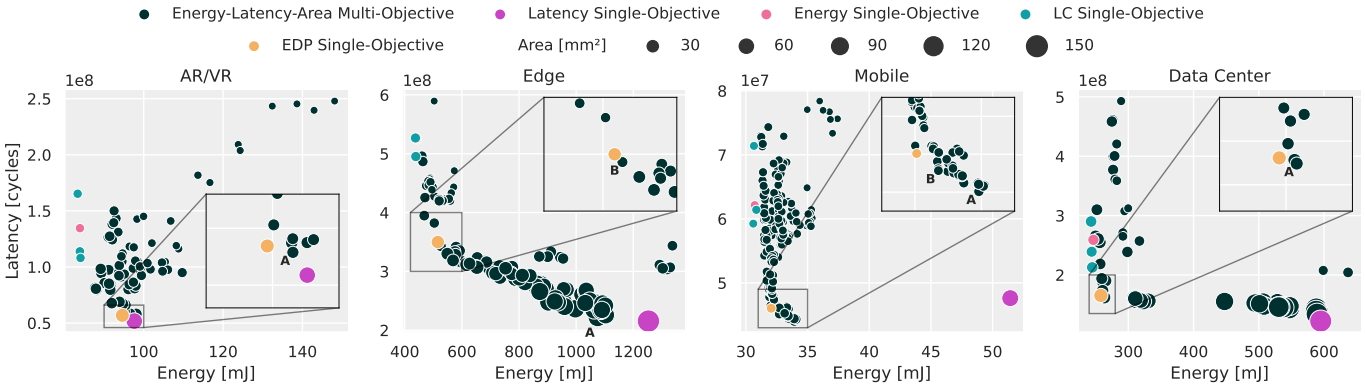


Figure 8: Comparison of individual solutions with mono-objective and Pareto solutions with multi-objective exploration.

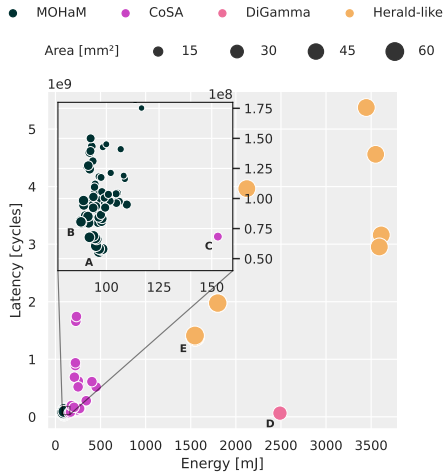


Figure 9: Comparison with state-of-the-art.

objective (yellow), multi-objective (black) design points A, B, B and A from the *AR/VR*, *Edge*, *Mobile* and *Data Center* workloads have 31.78%, 45.09%, 5.21% and 20.49% less area, respectively, while achieving the same EDP ($\pm 0.04\%$). Similarly, the other aggregated objectives, LCs, provide unbalanced solutions towards energy despite having different coefficients. This behaviour demonstrates that choosing coefficients in a weighted sum objective can be challenging and time-consuming when searching for compromise solutions. MOHaM supports distinct multi-objective exploration and provides a Pareto set of solutions (black). Across all workloads, it provides multiple competitive solutions with the lowest energy, latency and area, and exposes interesting trade-offs that can be leveraged. *The key takeaway from this experiment is that multi-objective exploration helps to identify the most suitable design for a specific multi-accelerator system.*

5.2.4 Comparison with State-of-the-Art

This experiment evaluates MOHaM-generated multi-accelerator system designs against three popular state-of-the-art, CoSA [12], DiGamma [19] and Herald [8]. The open-source implementations of CoSA and DiGamma are considered while Herald is implemented by faithfully following its evaluation settings. CoSA is a mono-objective, mapping-only optimisation that exploits Mixed-Integer Programming (MIP) to generate optimal mapping strategy without requiring iterative scheduling leveraging the Timeloop [17] cost model. DiGamma is a mono-objective, hardware-mapping co-optimisation that exploits GA (similar to MOHaM) for faster DSE leveraging the MAESTRO [38] cost model. Herald is a mono-objective, hardware-mapping co-optimisation that employs an exhaustive search and heuristic-based scheduling for DSE of multi-accelerator systems leveraging the MAESTRO cost model. Among the existing works on multi-accelerator systems, Herald is the only one targeting heterogeneous SAs, multi-DNN workloads and hardware-mapping co-optimisation (refer Table 1). However, there are a few important differences between Herald and MOHaM:

- Herald optimises the partitioning of the MAC budget among a fixed number of heterogeneous SAs. Whereas, MOHaM not only allows exploring the number of optimal SAs but also their buffer sizes.
- Herald uses a manual-tuned mapper that lacks flexibility and limits its scalability to diverse accelerator templates. Whereas, MOHaM exploits GA for its mapper that does not require tuning or modification with any accelerator template or workload variation.
- Herald provides Pareto optimal solutions for a mono-objective DSE. Whereas, MOHaM offers Pareto optimal solutions for a multi-objective DSE.

Herald originally employed the MAESTRO cost model, but to have a fair comparison with MOHaM, the Timeloop cost model is used here. Hence the resultant implementation is named *Herald-like* (and not exactly Herald). Additionally, the number of MACs and SAIs equal to that in the maximum configuration of MOHaM is considered for Herald-like. CoSA and DiGamma simulations are conducted considering flexible-dataflow architectures without considering the reconfiguration overheads. For CoSA, a Simba-like architecture is considered with the number of MACs equal to that in the maximum configuration of MOHaM. Its objective function is a linear combination of three components, *compute*, *traffic*, and *buffer utilisation*. Hence, it is run with different coefficients of these components for multiple solutions. For DiGamma, even though it originally employed the MAESTRO, an equivalent implementation with the Timeloop is also available and is used here. As DiGamma operates in a layer-wise manner, the results of all the layers from the workload are combined, and the area budget is set to the maximum configuration of MOHaM.

Figure 9 shows a subset of Pareto design points obtained by MOHaM along with the design points by CoSA, DiGamma and the best subset of Herald-like for the *AR/VR* workload. Table 5 presents configuration details of the specific design points (A through E) used for comparison. MOHaM's ability to generate Pareto design points offers

Table 5: Configuration of design points from Figure 9

Metrics	Design Points A	B	C	D	E
Latency [10^6 cycles]	55.76	80.59	68.33	64.12	1391.44
Memory Bound	64.69%	17.28%	100%	N.A.	35.43%
Energy [mJ]	96.53	87.93	152.91	2488.94	1547.88
Area [mm^2]	39.95	32.86	20.69	39.33	66.34
Compute Area	37.06%	36.25%	52.61%	17.25%	5.68%
PE Buf. Area	45.04%	43.12%	45.37%	71.85%	23.34%
Shared Buf. Area	17.90%	20.63%	2.02%	10.90%	70.98%
Total MACs	14336	16384	32768	32768	9984
Total SA Instances	7	8	1	1	3
Simba-like	7	5	1	0	2
Eyeriss-like	0	2	0	0	1
ShiDianNao-like	0	1	0	0	0

competitive alternatives to CoSA and DiGamma. For example, design point A from MOHaM reduces latency by 18.40% and energy by 36.87% compared to design point C from CoSA. Despite having a much higher number of MACs, CoSA's latency is higher than MOHaM. It can be attributed to CoSA lacking hardware optimisation and memory-bandwidth awareness. In fact, the whole execution is found to be memory-bounded (refer Table 5). The higher number of MACs (therefore PEs) experience increased communication overhead, resulting in higher energy consumption. As CoSA lacks hardware optimisation, its area is considered the minimum configuration of MOHaM to make a fair comparison. Hence, design point C from CoSA has the lowest area among all the design points from Table 5.

Similarly, design point A from MOHaM reduces latency by 13% and energy by 96.12% compared to design point D from DiGamma. Despite having a much higher number of MACs (same as CoSA) and hardware-mapping co-optimisation, DiGamma's latency is slightly higher than MOHaM. It can be attributed to DiGamma lacking reconfigurability of the architectural template. For example, DiGamma only has a two-level hierarchical architecture with PE buffers and shared buffers. Whereas MOHaM allows multi-level hierarchy for all the architectural templates supported by the Timeloop cost model. It is worth mentioning that no information about possible memory bottlenecks could be derived from DiGamma output logs. The energy consumption is very high in DiGamma due to large PE buffers. Table 5 shows that 71.85% of the DiGamma area is occupied by local PE buffers. They generate high-energy reads and writes, resulting in higher energy consumption.

When it comes to Herald-like, design point E is the best EDP solution. Nevertheless, design point A from MOHaM reduces latency by 96% and energy by 93.76% compared to design point E. Despite having a hardware-mapping co-optimisation with load-balance driven layer scheduling, Herald-like's latency and energy are much higher than MOHaM. It can be attributed to the random nature of Herald's hardware and mapping search coupled with the unavailability of multi-objective optimisation. Herald tried to improve latency by reducing memory-bound execution. Hence, it allocated large PE buffers and shared buffers, leaving only 5.68% compute area (refer Table 5). This action made the execution compute-bound, resulting in higher latency. Moreover, large PE buffers generate high-energy reads and writes, resulting in higher energy consumption. Furthermore, design point A from MOHaM also saves area by 39.78% compared to design point E from Herald-like.

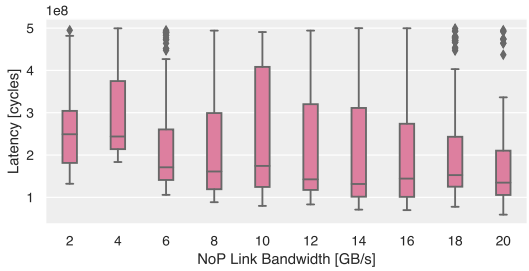


Figure 10: Latency against varying NoP link bandwidth.

Designers of multi-accelerator systems would benefit from having multiple competitive design points to choose from, especially when there are multiple objectives. However, a mono-objective optimisation aggregates everything into a single design point. This is acceptable as long as there are no conflicting objectives, which is rare. For example, when Herald's mono-objective optimisation aggregated latency and energy together into EDP, it is observed that neither the latency nor the energy is reduced. Herald's attempt to reduce latency by allocating large buffers backfired and resulted into increased latency. Moreover, large buffers are not desirable while attempting energy reduction. Hence, this conflicting scenario resulted in enormous energy consumption. On the other hand, a multi-objective optimisation offers multiple competitive design points while respecting all the objectives and not penalising anyone. For example, MOHaM's multi-objective optimisation allowed for a design point like A that could strike a balance between latency, energy, and area without severely penalising any of them. While there is no *one-size-fits-all*, based on the specific requirements, other design points can also be considered. For example, if energy efficiency is the preference, design point B from MOHaM improves energy by 42.50%, 96.47%, and 94.32% compared to CoSA, DiGamma, and Herald-like, respectively, at the cost of an increase in latency and area.

5.2.5 Sensitivity of NoP Link Bandwidth

This experiment evaluates the correlation of performance (latency) and NoP link bandwidth. Unlike state-of-the-art, where the communication cost is partially or entirely ignored, MOHaM implements it in detail. The communication model considers head/serialisation delay, routing paths, and bandwidth allocation on NoC and NoP links. Workload latency mainly depends on: (a) *SAI* hosting tile positions in NoP, (b) amount of data accessed from memory as per the layer mappings, and (c) NoP link bandwidth. MOHaM optimises *SAI* hosting tile positions to exploit maximum available bandwidth and avoid bottlenecks. Nevertheless, sometimes it becomes necessary to increase the bandwidth for the desired workload latency. Figure 10 shows the latency of Pareto solutions by MOHaM with varying NoP link bandwidth for the *AR/VR* workload. As expected, apart from some exceptions due to the stochastic nature of GAs, there is a trend of decreasing latency with increasing bandwidth. The biggest improvement is observed when the bandwidth increases from 2GB/s to 4GB/s. It is also worth observing that increasing the bandwidth beyond 16GB/s does not guarantee any improvement in latency. This anal-

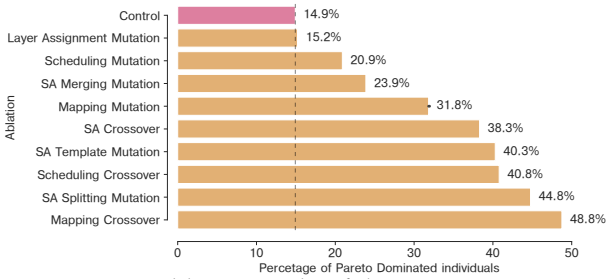


Figure 11: Ablation study of the custom GA operators.

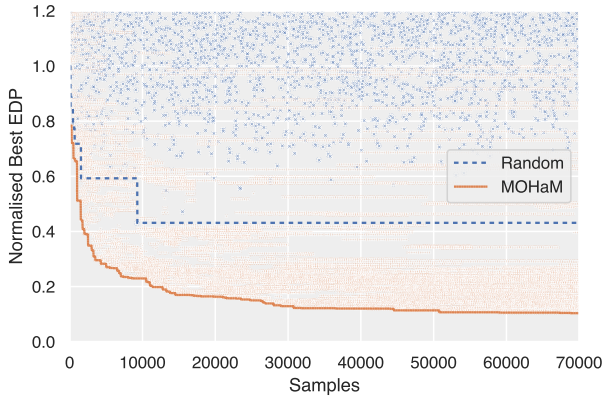


Figure 12: Sampling efficiency of MOHaM algorithm.

ysis using MOHaM can help reduce communication costs while maintaining the desired performance of the system.

5.2.6 Ablation Study

This experiment evaluates the effectiveness of the custom genetic operators shown in Figure 4. It compares the result of a complete MOHaM run with the results of runs, each with one operator disabled (ablated). Figure 11 shows the percentage of Pareto-dominated solutions when an operator is ablated from the baseline MOHaM configuration (refer Table 4). Due to multi-objective exploration, the results are obtained as follows: *(a)* MOHaM is run with the default configuration, and a baseline Pareto set of individuals is selected from the final population. *(b)* MOHaM is re-run with the same configuration to get a second Pareto set of individuals. *(c)* They are compared to identify how many individuals from the second set are Pareto-dominated by the individuals from the baseline set. This is found to be 14.9% and serves as the *Control* setting, as shown in Figure 11. *Control* serves as the threshold to compare ablation results. *(d)* MOHaM is run after disabling one operator to get a new Pareto set of individuals. *(e)* Steps *(c)* and *(d)* are repeated for all the operators, and the results are presented in Figure 11. A higher percentage of Pareto-dominated individuals indicate that the operator is effective and MOHaM's performance will deteriorate without it. All the operators perform better than the *Control* threshold, implying that each has some significance on MOHaM's performance.

5.2.7 Sampling Efficiency

This experiment evaluates the sampling efficiency of the GA-based MOHaM algorithm against a random exhaustive search. Several custom genetic operators are designed for a faster and sample-efficient DSE with MOHaM. Figure 12 shows the normalised EDP for $70,000$ solution samples for a DSE with random search and a DSE with MOHaM. The trend of solution discovery with the best EDP samples is highlighted with a curve. After these $70,000$ samples, MOHaM achieves $4.17\times$ better EDP compared to the random search. It is observed that some MOHaM solution samples with high EDP continue to be evaluated. This is due to the multi-objective exploration nature of the MOHaM algorithm. While the solution samples from random searches are scattered everywhere, most have poor EDP. The solution curve shows that a DSE with MOHaM will be at least orders of magnitude faster than with a random exhaustive search.

5.2.8 Execution Time

In the different experiment settings, MOHaM's execution time did not exceed eight hours, whereas CoSA took only about an hour. CoSA formulates the scheduling space into a MIP problem and solves it in one shot without requiring an iterative search. The scheduling solutions are then evaluated on the Timeloop cost model. Whereas MOHaM formulates the design space into a multi-objective GA problem and solves it by repeatedly querying the Timeloop. While the iterative nature of the GA allows for a more comprehensive exploration of the design space, it also requires more time to converge. Hence, most of MOHaM's execution time is spent repeatedly querying the Timeloop despite attempts to reduce the number of queries through caching. In general, the convergence time of an iterative search-based optimisation is directly proportional to the query response time of the cost model. Therefore, replacing the Timeloop with a faster cost model will ideally improve MOHaM's execution time.

It is important to note that CoSA is a compile-time mapping-only optimisation, so it only makes sense for CoSA to be executed as fast as possible. Whereas MOHaM is a design-time hardware-mapping co-optimisation, so it equally makes sense for MOHaM to be executed without worrying about time for a more comprehensive exploration. MOHaM is used offline and requires a one-time effort. Hence, MOHaM's absolute execution time is of no concern.

6 Related Works

DNN Models: Data center workloads are dominated by vision, language and recommendation-based DNN models [33][39]. Vision models are dominated by Convolution (CONV) with some MultiLayer Perceptron (MLP) and Fully Connected (FC) layers towards the end [27]. Language models are dominated by MLP, Recurrent Neural Network (RNN), embedding lookup and attention layers [40]. Whereas, recommendation models mainly consists of MLP, embedding lookup and attention layers [41].
Design Space Exploration: Hardware optimisations are used at design-time by ASICs or even at compile-time by FPGAs. Literature has multiple heuristic as well as ML-based hardware frameworks [14]. Mapping optimisations are used at compile-time or even at run-time by reconfigurable accelerators. Literature has mapping frameworks based on heuristics [28], random search [3], mixed integer programming [12], ML [42], etc. Hardware-mapping co-optimisations are used at design-time. Due to the huge cross-coupled search space, very few works have explored co-optimisation for single [19][29][42][28] and multi-accelerator systems [8][7]. **Multi-Tenancy:** Until recently, multi-tenancy has not been an important design choice for accelerators. Google TPU [2], Microsoft Brainwave [43],

etc., focused on running a single DNN model for maximum throughput. The popular MLPerf benchmark suite also focused on a single model for both, training [35] and inference [34]. Data centers are now employing multi-accelerator systems where throughput is not the only objective. Moreover, they have the natural ability to support diverse DNN models as each SA could favour a specific layer with its preferred dataflow. Hence, multi-tenancy has garnered significant attention recently [10][9][8][7][6][5].

7 Conclusion

This work presents MOHaM, a framework for multi-objective hardware-mapping co-optimisation for multi-DNN workloads on chiplet-based accelerators. The key takeaways are: (1) flexible dataflow with heterogeneous SAs increases the scalability of a multi-accelerator system toward diverse and emerging workloads, (2) hardware-mapping co-optimisation can support diverse workloads with the best overall performance in a multi-accelerator system, (3) multi-objective exploration helps identify the most suitable design for a specific multi-accelerator system, (4) a DSE with the proposed MOHaM framework will be at least orders of magnitude faster than with a random exhaustive search, and (5) replacing Timeloop with a faster cost model will improve MOHaM framework's execution time. A practical scenario where MOHaM could be employed is in designing a DNN-powered Advanced Driver-Assistance System (ADAS), like the ones used in Tesla cars [44]. Another practical scenario could be in designing DNN-based AR/VR hardware for the Metaverse [45]. Both of them have known workloads. Future work includes improving the exploration of the scheduling space to enhance the utilisation of the chiplets (SAs).

Acknowledgments

This work has been (partially) supported by MUR project ARS01_00592 reCITY Resilient City Everyday Revolution and by the Spoke 1 FutureHPC & BigData of the Italian Research Center on High-Performance Computing, Big Data and Quantum Computing (ICSC). Additionally, author A. Das gratefully acknowledges funding from the European Union's Horizon Europe research and innovation programme under grant agreement No 101042080 (WINC).

References

[1] "Google Edge TPUv1," https://cloud.google.com/edge-tpu/, 2018.
[2] (2021) Google Cloud TPUv4. https://cloud.google.com/tpu/.
[3] Y. S. Shao *et al.*, "Simba: Scaling Deep-Learning Inference with Multi-Chip-Module-Based Architecture," in *MICRO*, 2019.
[4] (2021) Cerebras CS-2. https://cerebras.net/system/.
[5] E. Baek *et al.*, "A Multi-Neural Network Acceleration Architecture," in *ISCA*, 2020.
[6] Y. Choi *et al.*, "PREMA: A Predictive Multi-Task Scheduling Algorithm for Preemptible Neural Processing Units," in *HPCA*, 2020.
[7] S. Ghodrati *et al.*, "Planaria: Dynamic Architecture Fission for Spatial Multi-Tenant Acceleration of Deep Neural Networks," in *MICRO*, 2020.
[8] H. Kwon *et al.*, "Heterogeneous Dataflow Accelerators for Multi-DNN Workloads," in *HPCA*, 2021.
[9] Z. Liu *et al.*, "VELTAIR: Towards High-Performance Multi-Tenant Deep Learning Services via Adaptive Compilation and Scheduling," in *ASPLOS*, 2022.
[10] S.-C. Kao *et al.*, "MAGMA: An Optimization Framework for Mapping Multiple DNNs on Multiple Acc. Cores," in *HPCA*, 2022.
[11] S. Zeng *et al.*, "Serving Multi-DNN Workloads on FPGAs: A Coordinated Arch, Sched, and Map Perspective," *IEEE TC*, 2022.
[12] Q. Huang *et al.*, "CoSA: Scheduling by Constrained Optimization for Spatial Accelerators," in *ISCA*, 2021.
[13] K. Hegde *et al.*, "Mind Mappings: Enabling Efficient Algorithm-Accelerator Mapping Space Search," in *ASPLOS*, 2021.
[14] S.-C. Kao *et al.*, "ConfuciuX: Autonomous Hardware Resource Assignment for DNN Accelerators using Reinforcement Learning," in *MICRO*, 2020.
[15] H. Kwon *et al.*, "Understanding Reuse, Performance, and Hardware Cost of DNN Dataflow: A Data-Centric Approach," in *MICRO*, 2019.
[16] K. Deb *et al.*, "A Fast and Elitist Multiobjective Genetic Algorithm: NSGA-II," *IEEE TEVC*, 2002.
[17] A. Parashar *et al.*, "Timeloop: A Systematic Approach to DNN Accelerator Evaluation," in *ISPASS*, 2019.
[18] Y. N. Wu *et al.*, "Accelergy: An architecture-level energy estimation methodology for accelerator designs," in *ICCAD*, 2019.
[19] S.-C. Kao *et al.*, "DiGamma: Domain-Aware Genetic Algorithm for HW-Mapping Co-Optimization for DNN Accelerators," in *DATE*, 2022.
[20] Y.-H. Chen *et al.*, "Eyeriss: An Energy-Efficient Reconfigurable Accelerator for Deep Conv Neural Networks," *IEEE JSSC*, 2016.
[21] E. Qin *et al.*, "SIGMA: A Sparse and Irregular GEMM Accelerator with Flexible Interconnects for DNN Training," in *HPCA*, 2020.
[22] Y.-H. Chen *et al.*, "Eyeriss v2: A Flexible Accelerator for Emerging Deep Neural Networks on Mobile Devices," *IEEE JETCAS*, 2019.
[23] (2018) NVDIA Deep Learning Accelerator (NVDLA). http://nvdla.org/.
[24] A. Parashar *et al.*, "SCNN: An Accelerator for Compressed-Sparse Convolutional Neural Networks," *ACM SIGARCH CAN*, 2017.
[25] S. Han *et al.*, "EIE: Efficient Inference Engine on Compressed Deep Neural Network," *ACM SIGARCH CAN*, 2016.
[26] Z. Du *et al.*, "ShiDianNao: Shifting Vision Processing Closer to the Sensor," in *ISCA*, 2015.
[27] K. He *et al.*, "Deep Residual Learning for Image Recognition," in *CVPR*, 2016.
[28] X. Yang *et al.*, "Interstellar: Using Halide's Scheduling Language to Analyze DNN Accelerators," in *ASPLOS*, 2020.
[29] E. Russo *et al.*, "MEDEA: A Multi-Objective Evolutionary Approach to DNN Hardware Mapping," in *DATE*, 2022.
[30] A. B. Kahn, "Topological Sorting of Large Networks," *CACM*, 1962.
[31] Y. Xu *et al.*, "A Genetic Algorithm for Task Scheduling on Heterogeneous Computing Systems using Multiple Priority Queues," *Elsevier Info Sci*, 2014.
[32] O. Roudenko *et al.*, "A Steady Performance Stopping Criterion for Pareto-based Evolutionary Algorithms," in *MOPGP*, 2004.
[33] M. Anderson *et al.*, "First-Generation Inference Accelerator Deployment at Facebook," *arXiv preprint arXiv:2107.04140*, 2021.
[34] V. J. Reddi *et al.*, "MLPerf Inference Benchmark," in *ISCA*, 2020.
[35] P. Mattson *et al.*, "MLPerf Training Benchmark," in *MLSys*, 2020.
[36] "Open Neural Network Exchange (ONNX)," https://github.com/onnx/onnx, 2017.
[37] A. E. Eiben *et al.*, "Parameter Tuning for Configuring and Analyzing Evolutionary Algorithms," *Elsevier Swa and Evol Comput*, 2011.
[38] H. Kwon *et al.*, "Maestro: A Data-Centric Approach to Understand Reuse, Performance, and Hardware Cost of DNN Mappings," *IEEE Micro*, 2020.
[39] D. Richins *et al.*, "Missing the Forest for the Trees: End-to-End AI Application Performance in Edge Data Centers," in *HPCA*, 2020.
[40] J. Devlin *et al.*, "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding," in *NAACL-HLT*, 2019.
[41] U. Gupta *et al.*, "DeepRecSys: A System for Optimizing End-to-End At-Scale Neural Recommendation Inference," in *ISCA*, 2020.
[42] Q. Xiao *et al.*, "HASCO: Towards Agile Hardware and Software Co-Design for Tensor Computation," in *ISCA*, 2021.
[43] J. Fowers *et al.*, "A Configurable Cloud-Scale DNN Processor for Real-Time AI," in *ISCA*, 2018.
[44] (2015) Tesla Autopilot. https://www.tesla.com/autopilot.
[45] (2021) Metaverse. https://about.meta.com/metaverse/.