Università degli Studi di Catania

Dipartimento di Ingegneria Elettrica Elettronica e Informatica

Dottorato in Ingegneria dei Sistemi,
Energetica, Informatica e delle Telecomunicazioni

Design Automation and Optimization for Post-Moore Computing Architectures:

hardware accelerators and quantum computers

---

phd thesis

Candidato
Enrico Russo

Tutor
Prof. Maurizio Palesi

Ciclo
XXXVIII

Coordinatore
Prof. Pietro Paolo Arena

---

30 Ottobre 2025

Acknowledgements

I would like to express my deepest gratitude to my supervisor, Prof. Maurizio Palesi, for his unwavering guidance, insightful advice, and constant support throughout my PhD.

I am also grateful to Prof. Paolo Arena, coordinator of the PhD School, and to all staff and faculty at the University of Catania for their support and for contributing to a productive research environment. Special thanks to colleagues and collaborators at the University of Catania who generously shared their expertise and practical suggestions: Vincenzo Catania, Davide Patti, Salvatore Monteleone, Giuseppe Ascia, Andrea Mineo, Daniela Panno, Luciano Miuccio, and Salvatore Riolo. I am particularly thankful to my colleagues Hamaad Rafique, Francesco G. Blanco, and Elio Vinciguerra for their friendship and help.

My sincere thanks go to Prof. Sergi Abadal for hosting me at UPC and for his support during my stay. I am thankful to Abhijit Das, Pol Puidgemont, and Axel Wassington for their collaboration, and to Pau, Mohammad, Sahar, Ama, Ausilia, Ylenia, and the other friends I had the pleasure of meeting in Barcelona.

For my time at ETH, I am grateful to Prof. Luca Benini for the opportunity and for his support, and to Alessio Burrello, Angelo Garofalo, Francesco Conti, and Daniele J. Pagliari for their supervision. I also thank Alessandro Ottaviano and Mohamed A. Hamdi for their collaboration, and all the people at the ETZ building. I am indebted to my flatmates and friends in Zurich — Martina, Maël, Angelina, Garance, Francesca, Martim, Milan, Django, Antonio Del Vecchio, and others — for their companionship.

Finally, I want to thank my family and lifelong friends for their unconditional love and support, without which this work would not have been possible.

Sommario

Automazione della Progettazione e Ottimizzazione per Architetture di Calcolo Post-Moore: acceleratori hardware e computer quantistici

La crescita incessante della densità dei transistor, resa possibile dallo scaling di Dennard e osservata da Moore, si è stabilizzata negli ultimi due decenni, portando a profondi cambiamenti nell'architettura dei computer. I fenomeni conseguenti, noti come "power wall" e "dark silicon", hanno spinto il settore lontano dalle CPU omogenee scalate in frequenza verso Sistemi su Chip (SoC) eterogenei che integrano acceleratori specifici per dominio (DSA) e, sempre più spesso, verso nuovi paradigmi come il calcolo quantistico e neuromorfico. Parallelamente, l'esplosiva crescita dei modelli di deep learning (DL) e intelligenza artificiale (AI)—caratterizzata da drastici aumenti nei parametri, nella domanda computazionale e nel consumo energetico—ha amplificato l'urgenza di una specializzazione e ottimizzazione hardware efficiente sia nei data center che negli ambienti edge. Il calcolo quantistico, sebbene ancora agli inizi, sta emergendo come acceleratore specializzato con vincoli e opportunità unici, complicando ulteriormente il panorama della co-progettazione hardware–software.

Questa tesi affronta le sfide convergenti delle moderne architetture eterogenee sviluppando nuovi metodi per l'esplorazione dello spazio di progetto (DSE), il mapping, la schedulazione e l'ottimizzazione cross-layer sia per acceleratori AI che per sistemi quantistici. I contributi principali includono: (i) un sistema di mapping basato su programmazione matematica per il deployment efficiente di workload tensoriali su architetture spaziali; (ii) un framework DSE alimentato da algoritmi genetici per l'ottimizzazione di sistemi multi-acceleratore e workload multi-DNN; (iii) tecniche basate su reinforcement learning e dati per la schedulazione online e l'ottimizzazione del dataflow in acceleratori AI multi-tenant e di graph neural network; e (iv) approcci innovativi per il mapping e il routing dei qubit in architetture quantistiche modulari, minimizzando la comunicazione e la contesa delle risorse. Le metodologie presentate contribuiscono alla co-progettazione automatizzata, scalabile ed energeticamente efficiente delle piattaforme di calcolo di nuova generazione.

Abstract

Design Automation and Optimization for Post-Moore Computing Architectures: hardware accelerators and quantum computers

The relentless scaling of transistor density, once epitomized by Moore's Law and enabled by Dennard scaling, has plateaued over the past two decades, ushering in profound shifts in computer architecture. The resulting "power wall" and "dark silicon" phenomena have propelled the field away from homogeneous, frequency-scaled CPUs toward heterogeneous Systems-on-Chip (SoCs) that integrate domain-specific accelerators (DSAs) and, increasingly, novel paradigms such as quantum and neuromorphic computing. Simultaneously, the explosive growth of deep learning (DL) and artificial intelligence (AI) models, characterized by dramatic increases in parameter counts, computational demand, and energy consumption, has amplified the urgency for efficient hardware specialization and optimization across both data center and edge environments. Quantum computing, though still nascent, is emerging as a specialized accelerator with unique constraints and opportunities, further complicating the hardware–software co-design landscape.

This thesis addresses the converging challenges of modern heterogeneous architectures by developing new methods in design space exploration (DSE), mapping, scheduling, and cross-layer optimization for both AI accelerators and quantum systems. Key contributions include: (i) a mathematical programming-based mapper for efficient deployment of tensor workloads on spatial architectures; (ii) a genetic algorithm-powered DSE framework for optimizing multi-accelerator systems and multi-DNN workloads; (iii) reinforcement learning and data-driven techniques for online scheduling and dataflow optimization in multi-tenant AI inference and graph neural network accelerators; and (iv) novel approaches to qubit mapping and routing in modular quantum architectures, minimizing communication and resource contention. The methodologies presented here contribute to automated, scalable, and energy-efficient co-design of next-generation computing platforms.

Contents

Chapter 1

Overview

Since Gordon Moore's 1965 empirical observation that the number of transistors on an integrated circuit would approximately double every 18 months [148], the semiconductor industry has enjoyed decades of exponential scaling (Figure 1.1). For nearly half a century this trend underpinned continual performance improvements at roughly constant cost per function. Yet, in the past two decades, the historical cadence of density scaling has markedly slowed down [91].

Concurrently, the industry encountered the "power wall" [90]: as transistors shrank, leakage currents and switching activity drove up power consumption and heat density, constraining further increases in clock frequency. Historically, Dennard scaling posited that voltage and current could scale with feature size such that power density remained roughly constant even as transistor counts increased [53]. However, this assumption ceased to hold more than a decade ago [91], primarily due to leakage and variability in deeply scaled CMOS. The erosion of both Moore's and Dennard's scaling compounds energy efficiency challenges: frequency scaling has stalled, and raw transistor abundance no longer translates automatically into proportional performance gains.

In response, architectural innovation shifted from single, ever-faster cores to multi-core designs, aggregating several general-purpose cores on a single die to exploit thread- and task-level parallelism. While effective for a subset of workloads, this approach faces fundamental limits: (i) Amdahl's Law constrains parallel speedup for partially serial applications; (ii) scaling core counts increases on-chip interconnect, cache coherence, and memory hierarchy overheads; (iii) software engineering complexity and portability concerns escalate; and (iv) thermal and power delivery constraints impose practical ceilings, leading to "dark silicon", i.e., regions that must remain power-gated to satisfy energy and thermal envelopes [59, 64, 222].

As the bottom (device-level) scaling frontier tightens, performance and efficiency opportunities migrate upward in the computing stack [132], echoing Feynman's early visionary perspective on technological miniaturization [69]. A principal vector for continued advancement is specialization: heterogeneous Systems-on-Chip (SoCs) integrate

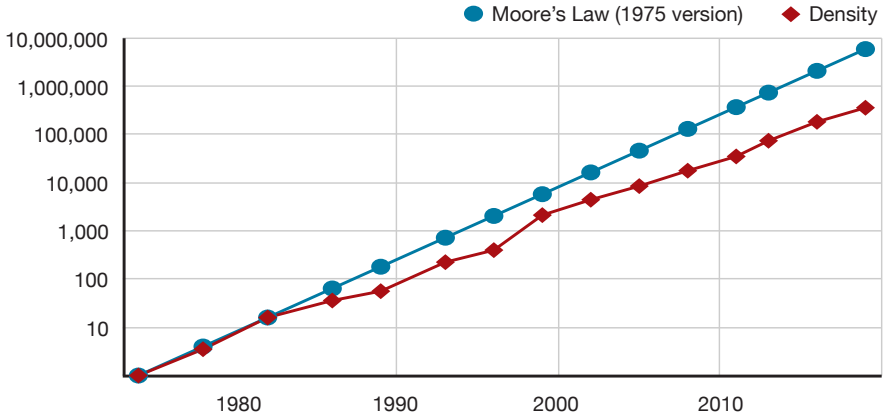


FIGURE 1.1: Intel microprocessor transistor counts compared to Moore's Law projection [91].

Domain-Specific Accelerators (DSAs) alongside general-purpose CPUs. Unlike general-purpose cores, DSAs eliminate the instruction, control-flow, and memory hierarchy overheads not essential to their target domain, achieving superior performance-per-watt for selected workloads [47]. For modern Artificial Intelligence (AI) and Machine Learning (ML) applications, this paradigm has produced specialized Neural Processing Units (NPUs), systolic tensor cores, and reconfigurable dataflow fabrics.

In parallel, radically different computational paradigms are maturing. Quantum computing aims to accelerate specific classes of problems (e.g., certain optimization, simulation, and algebraic tasks) by exploiting superposition and entanglement [18, 20]. Neuromorphic architectures [199], spintronic devices [214], and emerging two-dimensional or graphene-like materials [85] explore alternative physical substrates with potentially favorable energy or functional characteristics. These paradigms are not mutually exclusive: future heterogeneous platforms may integrate classical CPUs, GPUs, FPGAs, AI accelerators, and quantum processors, orchestrated to exploit each technology's niche advantages.

Viewed through the lens of heterogeneity, a quantum processing unit (QPU) can itself be treated as a domain-specific accelerator with a fundamentally distinct execution and error model. Figure 1.2 would illustrate such a composite system comprising CPUs, GPUs, NPUs, FPGAs, and quantum devices coordinated via a unified runtime or compilation framework.

However, two intertwined challenges arise:

1. **Design Space Explosion for Accelerators.** Microarchitectural design of AI accelerators (e.g., dataflow scheduling, memory hierarchy sizing, on-chip network topology, quantization support, tensor tiling strategies) spans a vast combinatorial

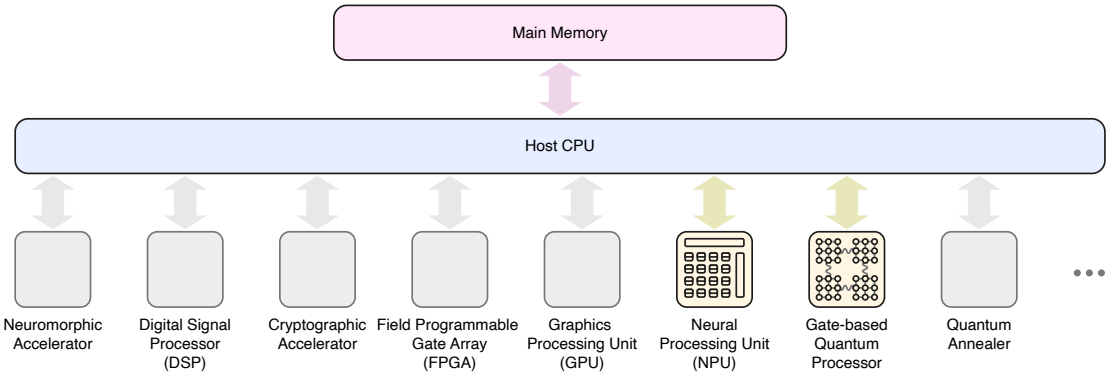


FIGURE 1.2: Conceptual heterogeneous system architecture [18]. Highlighted accelerators are in the scope of this dissertation.

design space. Exhaustive exploration is infeasible; naive heuristic-driven pruning risks suboptimal energy-performance trade-offs.

2. **Compilation and Mapping Complexity.** Efficiently deploying workloads (e.g., classical ML graphs on spatial dataflow fabrics or quantum circuits on noisy constrained hardware) requires advanced compilation pipelines. These must optimize for latency, throughput, energy, reliability, and resource contention while respecting device-specific constraints (e.g., qubit connectivity, error rates, or on-chip buffer capacities).

Bridging these gaps requires automated Design Space Exploration (DSE) methodologies coupled with specialized compilation and scheduling frameworks. For AI accelerators, such frameworks must jointly reason about algorithmic transformations (e.g., operator fusion, tiling, sparsity exploitation) and hardware configuration parameters. For quantum systems, compilers qubit allocation, routing, error-aware scheduling, and resource balancing on current constrained architectures.

1.1 The Contemporary Surge of Artificial Intelligence

Deep learning (DL), and in particular deep neural networks (DNNs), have become the dominant paradigm of contemporary AI because of their capacity to learn rich hierarchical representations from large-scale heterogeneous data. Their impact now spans computer vision [181], large-scale natural language and multimodal modeling [2, 227], audio and speech synthesis [216], recommender systems at web scale [153], autonomous driving perception–and–planning stacks [82], robotics and visuomotor control [134], medical imaging and clinical decision support [12], and drug discovery pipelines [163], among

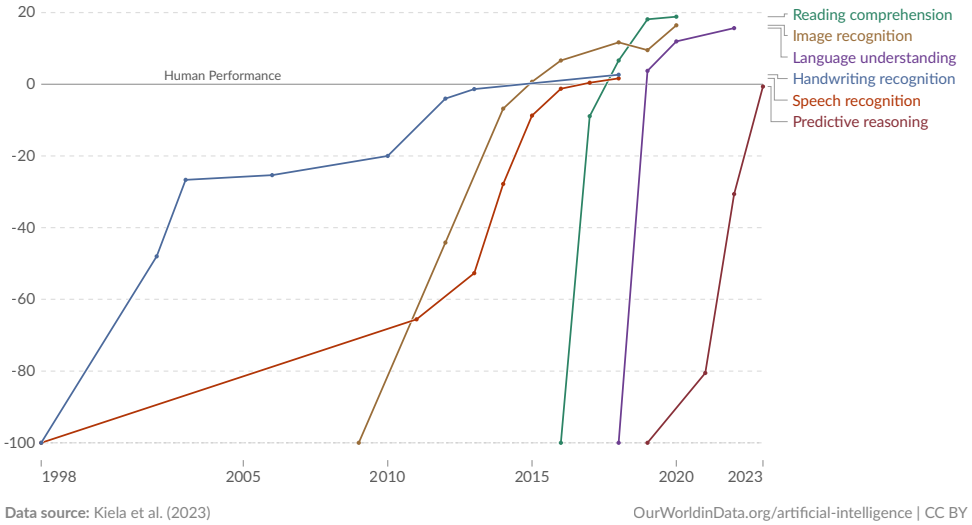


FIGURE 1.3: Test scores of AI systems on various capabilities relative to human performance

many others. The emergence of large "foundation" models pre-trained on trillions of tokens and adapted by fine-tuning or prompting has further accelerated the capability [177]. Figure 1.3 shows the trend in the increase in deep learning capabilities across different tasks and highlights that in many domains super-human performance has already been achieved.

A defining feature of recent progress is the steep acceleration of computational demand. Model parameter counts, dataset token volumes, and effective training FLOPS have followed empirically observed scaling laws [113]. Figure 1.4 illustrates the (approximately) exponential growth of parameter counts in state-of-the-art models originating from both academia and industry, spanning early convolutional and sequence models through billion– to trillion–parameter transformer-based architectures. This relentless upward trend is accompanied by a corresponding increase in peak memory footprint, compute operations, communication volume (in distributed data / tensor parallel regimes), and energy consumption per training run [42]. Figure 1.5 illustrates the growing number of floating-point operations required to train contemporary deep learning models, showcasing a four-fold increase each year on average.

The macro-level sustainability implications are non-trivial. For data centres, projections indicate a substantial energy consumption increase driven in large part by AI workloads. As shown in Figure 1.6, it is envisaged that data centre electricity demand could more than double by 2030 [103], underscoring the urgency of holistic efficiency improvements and carbon-aware optimizations [169].

There is also a growing interest in relocating parts of the AI computation continuum

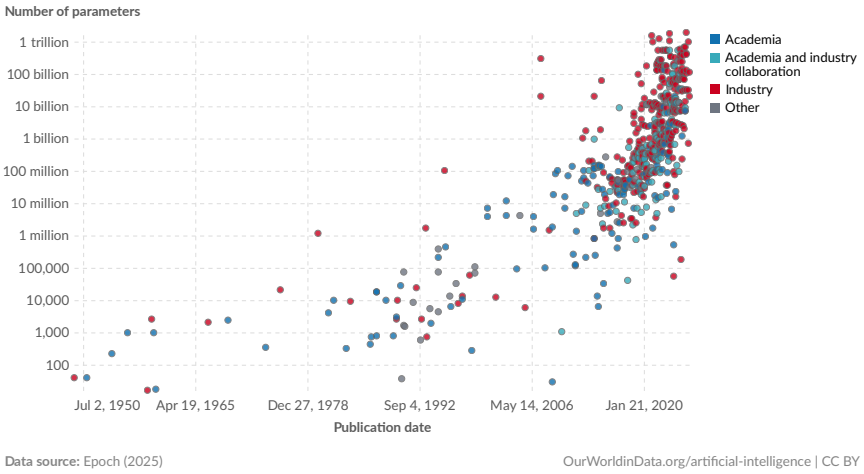


FIGURE 1.4: Number of parameters in notable artificial intelligence systems

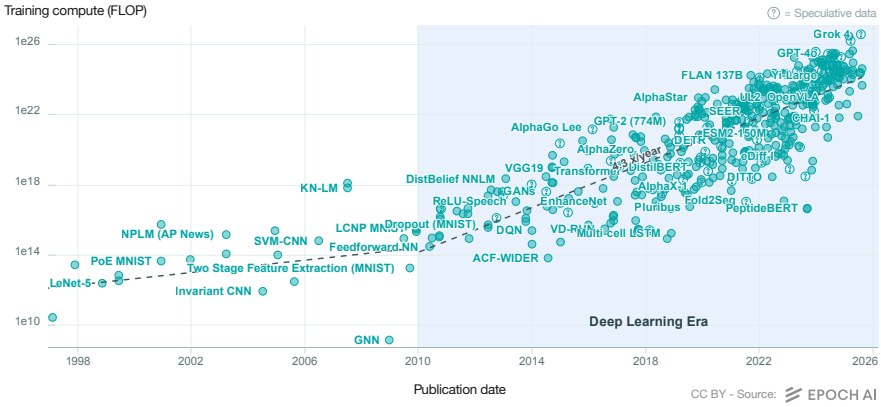


FIGURE 1.5: Estimated number of floating-point operations needed to train notable deep learning models.

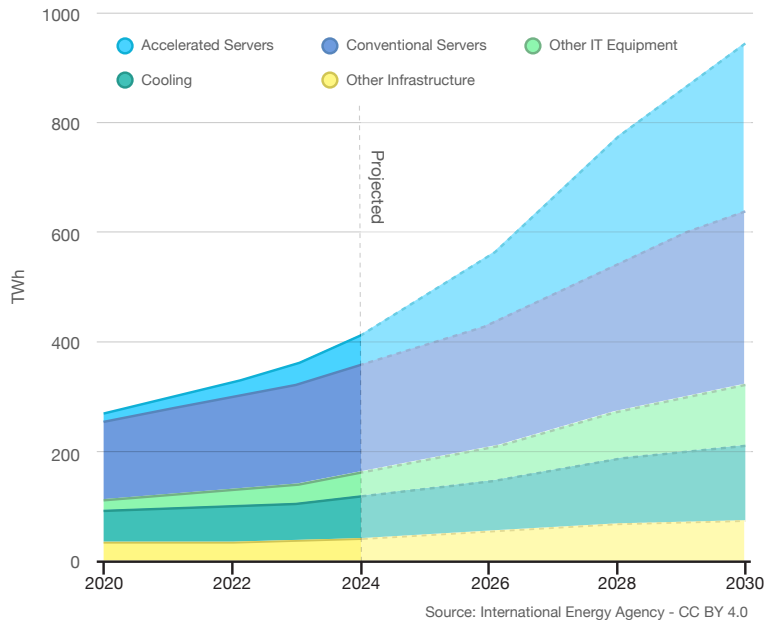


FIGURE 1.6: Global data centre electricity consumption projection by equipment [103].

from centralized hyperscale clouds to resource-constrained edge, embedded, and mobile Internet-of-Things (IoT) platforms. In fact, local execution offers: (i) reduced end-to-end latency by eliminating internet round-trips, which might be crucial for latency-sensitive applications such as augmented reality and autonomous driving; (ii) improved privacy by retaining raw sensor/user data on-device; (iii) lower backbone network traffic; and (iv) resilience in bandwidth-limited or intermittently connected environments. Yet embedded and mobile devices remain constrained in compute throughput, on-chip and off-chip memory capacity and bandwidth, and energy (battery or harvested).

Therefore, efficiently mapping complex DNNs and optimally designing custom accelerators is a central research challenge for both data centers and embedded platforms [211].

1.2 The Early Era of Quantum Computing

Quantum computing has developed from a primarily theoretical concept into an experimental science, with prototype hardware already demonstrating computational behaviors beyond the capabilities of classical systems. The promise of quantum computing is grounded in its controlled use of superposition, entanglement, and interference, that is, mechanisms enabling speedups for specific classes of problems [155]. Notable examples

include Shor's algorithm, which challenges the security of widely-used public-key cryptosystems [202]; Grover's algorithm, which provides quadratic speedup for unstructured search [83]; quantum simulation of Hamiltonians with applications in chemistry, materials science, and catalysis [46]; and variational or amplitude-based approaches that hold promise in optimization, machine learning, and complex logistical analyses [31]. Even within the era of noisy intermediate-scale quantum (NISQ) devices, hybrid quantum-classical workflows, where quantum circuits serve as domain-specific accelerators within classical routines, are already being explored [28, 143].

Despite this potential, significant challenges remain at every level, from physical qubits up to the algorithmic layer. Qubits are inherently fragile, with decoherence, gate infidelity, crosstalk, and leakage all limiting the achievable depth and reliability of quantum circuits [172]. As error rates grow with both circuit depth and width, large-scale quantum computation depends on quantum error correction (QEC), which substantially increases resource requirements, often by necessitating thousands of physical qubits per logical qubit depending on the chosen code and target error rate [80]. As a result, the raw number of qubits does not directly reflect true computational power, and can be misleading as an indicator of progress toward practical utility. Additional bottlenecks arise from material defects, fabrication variability, frequency crowding in superconducting devices, mode management in ion traps, photon loss in photonic platforms, and the scaling of control electronics. Further engineering challenges include managing thermal loads and wiring at cryogenic temperatures, as well as the complexity of calibration and drift mitigation. On the algorithmic front, reducing circuit depth through compilation optimizations, approximate gate synthesis, gate cancellation, and layout-aware scheduling is essential for operating within limited coherence times until large-scale error correction becomes viable [106, 207].

A central technical obstacle in scaling quantum computers toward utility-scale is the challenge of routing and qubit mapping, that is, translating idealized quantum circuits into device-specific instructions [225]. Most hardware architectures support only sparse, locality-constrained connectivity; for example, nearest-neighbor couplings on two-dimensional lattices (superconducting qubits), limited long-range interactions (neutral atoms), chain or bandwidth-limited connections (trapped ions), or probabilistic photonic links. While high-level quantum algorithms often assume full connectivity, mapping them onto hardware with restricted topologies requires the insertion of SWAP operations or teleportation-like steps to bring qubits into proximity for required interactions [43]. Each SWAP operation increases both latency and error probability, inflating the circuit depth and reducing the likelihood of successful execution. Finding optimal qubit placements is computationally difficult and is related to graph embedding and token swapping problems [25], which leads compilers to use heuristic methods such as lookahead search, temporal partitioning, reinforcement learning, SAT/SMT formulations, or mixed integer programming for tractable cases. In modular or distributed

quantum systems, routing must also address the generation, purification, and buffering of entanglement across quantum networks, where the stochastic nature of link formation calls for adaptive and efficient scheduling [66]. Progress in routing and mapping algorithms directly expands the set of circuits that can be implemented on existing hardware, effectively increasing computational capacity and advancing the field toward utility scale quantum computing without any physical hardware changes.

Quantum computing is therefore at a critical juncture, where realizing its theoretical advantages requires aligning algorithmic ambitions with the realities of engineering and hardware. Major industry players such as Google and IBM have published detailed technology roadmaps [76, 100], outlining their strategies to reach utility scale quantum computing, defined as the threshold at which quantum devices can solve useful problems beyond the reach of classical supercomputers with practical reliability. These roadmaps emphasize not only aggressive increases in qubit numbers, but also improvements in fidelity, error correction, system integration, and software-toolchain co-design. Future progress will depend on concurrent advances in reducing gate errors, improving device fabrication and control, refining error correction protocols, and, crucially, developing compilation pipelines that are both topology- and noise-aware. Effective routing and mapping are not peripheral optimizations, but central engineering challenges that enable incremental hardware improvements to translate into much greater computational capabilities. Addressing these issues is essential for accelerating the transition to practically and economically meaningful quantum advantage at utility scale. A primary focus of this dissertation is advancing the state of the art in quantum routing and mapping optimization.

1.3 Scope and Contributions

This thesis is motivated by the convergence of these trends: (i) the diminishing returns of traditional scaling, (ii) the strategic shift toward heterogeneity and specialization for AI and other domains, and (iii) the emergence of quantum computing as an accelerator-like component of future computing stacks. Together, these trends demand new methodologies that span (a) architecture-level DSE, (b) mapping and compilation techniques that bridge application abstractions and hardware substrates, (c) runtime or compile-time scheduling for efficient resource utilization.

Figure 1.7 provides a high-level cartography of the publications produced during doctoral work. Each node denotes a peer-reviewed publication. Nodes are positioned within and at the intersections of four thematic sets: Architecture, Mapping, Scheduling, and Optimization. Colored nodes correspond to contributions that are integrated, fully or in adapted form, into the core narrative of the dissertation; faded nodes denote ancillary work that falls outside the thesis' unifying through-line and are therefore only cited contextually or omitted for focus. Overlaps highlight cross-layer contributions (e.g.,

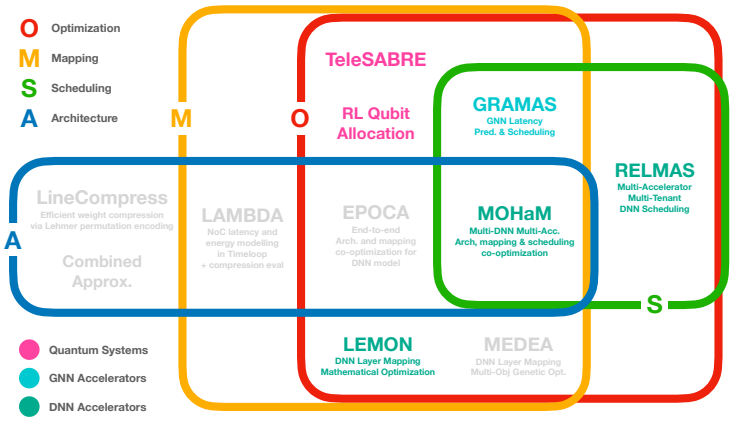


FIGURE 1.7: Conceptual "metro map" of the doctoral publications organized as a four-set Venn diagram: Architecture, Mapping, Scheduling, and Optimization. Each node represents a publication. Colored nodes are incorporated into the thesis' main argument; gray nodes denote peripheral or supplemental work not elaborated in detail. Nodes in overlapping regions indicate cross-layer contributions.

a scheduling method that depends on an architectural analytical model; or a mapping heuristic that simultaneously performs cost-driven optimization).

The thesis as a whole is divided into two parts: Part I addresses DSE, scheduling optimization, and mapping techniques for AI domain-specific accelerators; Part II extends the methodological backbone, particularly mapping, scheduling, and cross-layer optimization principles, to emerging quantum architectures.

Contributions

In summary, this thesis provides the following contributions:

- A mathematical programming based mapper designed for tensor workloads on spatial architectures, taking into account memory access costs and bandwidths (Chapter 3). This tool facilitates the mapping of DNN operations on hardware accelerators both temporally and spatially, aiming to minimize energy consumption and latency.

- A DSE framework utilizing genetic algorithms to concurrently optimize architecture, mapping and scheduling in multi-accelerator systems targeting multi-DNN workloads (Chapter 4). Given a set of architectural templates for hardware accelerators and a set of DNN models, this tool aims to obtain the Pareto set of

multi-accelerator architecture design, layer scheduling/allocation and layer mapping considering area, latency and energy as objective metrics.

- A reinforcement learning approach to online scheduling in multi-tenant multi-accelerator AI inference systems (Chapter 5).
- A data-driven approach to online dataflow and scheduling optimization in reconfigurable graph neural networks inference systems (Chapter 6).
- A reinforcement learning approach to qubit mapping in modular quantum systems aimed at minimizing inter-core quantum state transfers (Chapter 8).
- An heuristic for intra-core and inter-core qubit routing in teleport-interconnected multi-core quantum systems (Chapter 9).

The research presented in Chapter 4 was conducted in collaboration with Abhijit Das (associated with Universitat Politècnica de Catalunya, Spain). The work described in Chapter 5 was developed in collaboration with Francesco Giulio Blanco (affiliated with University of Catania, Italy), while the study in Chapter 6 was carried out in collaboration with Pol Puidgemont Plana and Axel Wassington (associated with Universitat Politècnica de Catalunya, Spain).

Part I

Hardware Accelerators for Artificial Intelligence

Chapter 2

Background on Domain-Specific Accelerators for AI

The rapid advancement of deep learning and in particular DNNs has transformed artificial intelligence into a pervasive force across domains ranging from computer vision and language understanding to autonomous vehicles and healthcare. As highlighted in the introduction, the astonishing capabilities of modern DNNs stem from their ability to learn hierarchical representations from large, diverse datasets, often surpassing human-level performance in complex real-world tasks. This progress, however, is inseparable from the underlying computational infrastructure: the exponential growth in model size, dataset scale, and required computation has outpaced the scaling afforded by traditional general-purpose processors.

The proliferation of AI workloads, spanning resource-constrained edge devices to hyperscale data centers, has created an acute need for specialized hardware capable of delivering high throughput and energy efficiency under stringent power, area, and latency constraints [3, 114, 240]. Yet, embedded platforms, such as those powering mobile and IoT devices, face severe limitations in power budget and silicon area. Offloading inference to the cloud can alleviate device constraints, but it often conflicts with application requirements for privacy, security, or real-time responsiveness. In scenarios where data locality, privacy, or ultra-low latency is paramount, on-device or edge processing becomes essential [241]. Even within data centers, the relentless growth in AI inference and

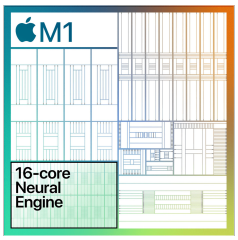


FIGURE 2.1: Neural Engine in Apple M1 SoC

training workloads is driving demand for ever greater energy efficiency and performance to meet service-level objectives and sustainability targets [73, 158, 238].

Against the backdrop of the slowing of Moore's Law and the breakdown of Dennard scaling, architectural innovation has shifted decisively toward specialization. Domain-Specific Accelerators (DSAs), custom hardware blocks tailored for particular classes of algorithms, have emerged as a central vector for continued improvements in performance-per-watt [231]. In the context of DNNs, Neural Processing Units (NPUs) and related engines are now ubiquitous, both as standalone chips and as integrated components within heterogeneous System-on-Chip (SoC) platforms (Figure 2.1). These SoCs combine general-purpose CPUs with a diverse array of specialized accelerators—GPUs, Image Signal Processors (ISPs), Digital Signal Processors (DSPs), encryptors, video encoders, and DNN accelerators—reflecting the heterogeneous computing paradigm [6, 236].

Modern DNN accelerators are architected to exploit the massive parallelism and regular structure of neural network computations [219]. Typically, they feature large arrays of tightly coupled compute and storage units organized in two-dimensional meshes, optimized for the matrix and tensor operations at the heart of deep learning. Architectural mechanisms such as high-bandwidth memory (HBM), on-chip SRAM scratchpads, network-on-chip (NoC) fabrics, and sophisticated data reuse strategies are essential to minimize data movement—often the dominant source of energy consumption [39]. The design of dataflows, i.e., strategies for orchestrating the movement and reuse of weights, activations, and partial sums, plays a critical role in maximizing computational efficiency. Weight-stationary [157], output-stationary [57], row-stationary [35], and reconfigurable dataflows [128, 129, 141] each embody specific trade-offs between memory footprint, bandwidth, and energy.

This landscape gives rise to a rich diversity of DNN accelerator architectures, each making distinct choices about parallelism granularity, memory hierarchy, interconnect topology, and supported dataflows. Notably, the open-source NVDLA [157] exemplifies the weight-stationary approach for automotive and embedded AI, while Simba [200] extends this to chiplet-based multi-chip modules interconnected by a network-on-package (NoP). The Eyeriss [35] and ShiDianNao [57] accelerators, among others, have introduced new dataflow paradigms and highlighted the energy and performance trade-offs inherent in DNN mapping.

In summary, the shift toward domain-specific, heterogeneous architectures has propelled the development of highly optimized DNN accelerators tailored to both edge and cloud deployments. The remainder of this chapter is organized as follows: Section 2.1 surveys representative AI workloads accelerated by DSAs; Section 2.2 details the architectural principles and mapping techniques that underpin efficient DNN acceleration, including the loop nest notation for computation mapping; and Section 2.3 reviews the

landscape of simulation and design space exploration tools enabling the systematic co-design of algorithms and accelerator hardware.

2.1 Typical AI Workloads

This section characterises representative deep learning workload classes that drive DSE for DSAs. We emphasise their dominant computational kernels, data reuse and memory access patterns, and structural regularity or irregularity—factors that determine attainable parallelism, on-chip buffering efficiency, and interconnect pressure. The focus is on inference acceleration.

Core Computational Motifs Across contemporary model families a small, recurrent set of primitive operations accounts for most cycles and energy [60]: dense general matrix multiplications (GEMMs) in fully connected (FC) layers, linear projections and transformer feed-forward blocks; multi-dimensional convolutions in vision models; sparse–dense matrix products in graph neural networks (GNNs) and pruned networks; attention score and value projections (again GEMMs) plus softmax reductions in transformers; elementwise operations (activations, bias additions, residual adds, normalisations); reductions (softmax, layer, batch or group normalisation); and memory-intensive embedding or gather/scatter phases. Although expressible largely as tensor contractions plus pointwise transforms, each workload class stresses different shape regimes (aspect ratios, batch sizes), reuse distances and sparsity structures, which in turn shape efficient accelerator dataflows.

Multi-Layer Perceptrons (MLPs) An MLP (Fig. 2.2) stacks fully connected layers made of single perceptrons (Fig. 2.3) with non-linear activations:

$$\mathbf{y} = f(\mathbf{W}\mathbf{x} + \mathbf{b}),$$

where f is typically ReLU, sigmoid, tanh or a modern variant. Inference time is dominated by dense GEMMs (Fig. 2.4) with high arithmetic intensity and regular memory strides. Weight reuse across batched inputs and activation reuse within tiles make blocking and prefetching effective. Model compression (pruning, low-rank factorisation, quantisation / weight sharing) reduces both memory footprint and external bandwidth, exposing opportunities for sparsity-aware execution units.

Convolutional Neural Networks (CNNs) CNNs exploit local spatial correlation and weight sharing over grid-structured data (e.g. images) [131, 211] namely feature maps. As shown in Fig. 2.5, a 2D convolution layer applies K learned kernels of size $R \times S$ across C input channels to produce K output channels. Stride controls filter shift

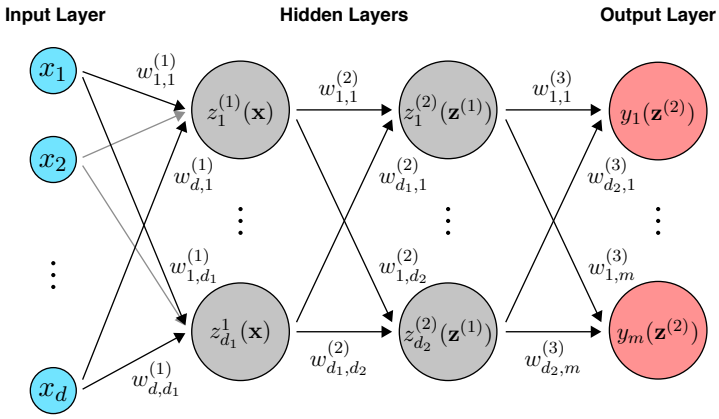


FIGURE 2.2: Illustrative feed-forward artificial neural network (MLP) showing stacked fully-connected layers with activations.

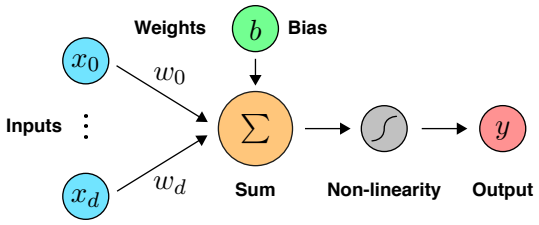


FIGURE 2.3: Perceptron diagram

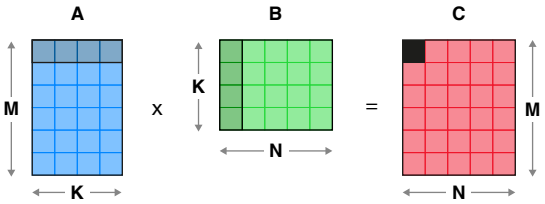


FIGURE 2.4: General matrix multiplication $\mathbf{AB} = \mathbf{C}$ where $\mathbf{A} \in \mathrm{R}^{M \times K}$, $\mathbf{B} \in \mathrm{R}^{K \times N}$ and $\mathbf{C} \in \mathrm{R}^{M \times N}$.

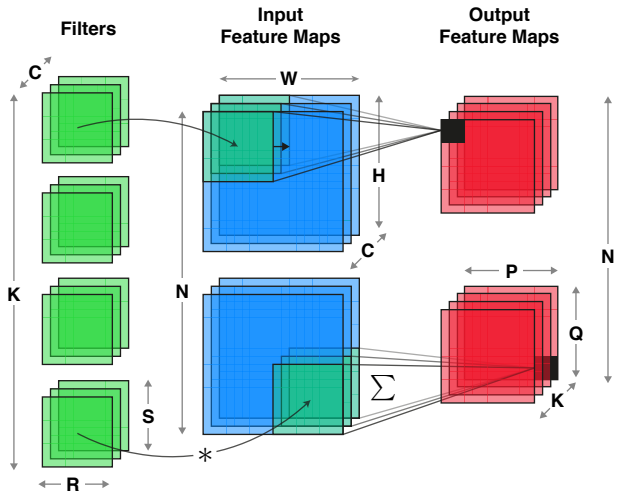


FIGURE 2.5: Tensor dimensions in a batched convolution layer (input, filter bank, output).

steps and downsampling of the feature map; dilation can enlarge receptive fields; logical padding preserves or alters spatial extent.

Each output activation draws from a localized receptive field, yielding structured reuse: inputs are reused across overlapping windows; each kernel weight is reused over all spatial positions and batch elements; partial sums accumulate across the channel dimension. These multi-level reuse opportunities motivate hardware dataflows (weight-stationary, output-stationary, row-stationary) that keep frequently reused operands on chip to reduce DRAM energy. Depthwise and pointwise convolutions decompose standard convolutions to lower parameter count and multiply–accumulate (MAC) density, shifting the balance toward memory bandwidth.

Recurrent Neural Networks (RNNs) RNNs and gated variants (LSTM, GRU) model temporal dependencies by maintaining a hidden state updated at each timestep through several small to medium GEMMs followed by elementwise gates. Their principal hardware challenges versus CNNs are reduced parallelism (strict inter-timestep dependence), smaller matrix shapes (risking underutilisation of wide systolic arrays), and the need for efficient batching or multi-stream processing to amortise overhead. High weight reuse across timesteps, however, allows key matrices to remain resident in on-chip SRAM for low-latency access.

Transformers Transformers [217] eliminate explicit recurrence via self-attention, enabling parallel processing over all tokens of a sequence (bounded by quadratic attention

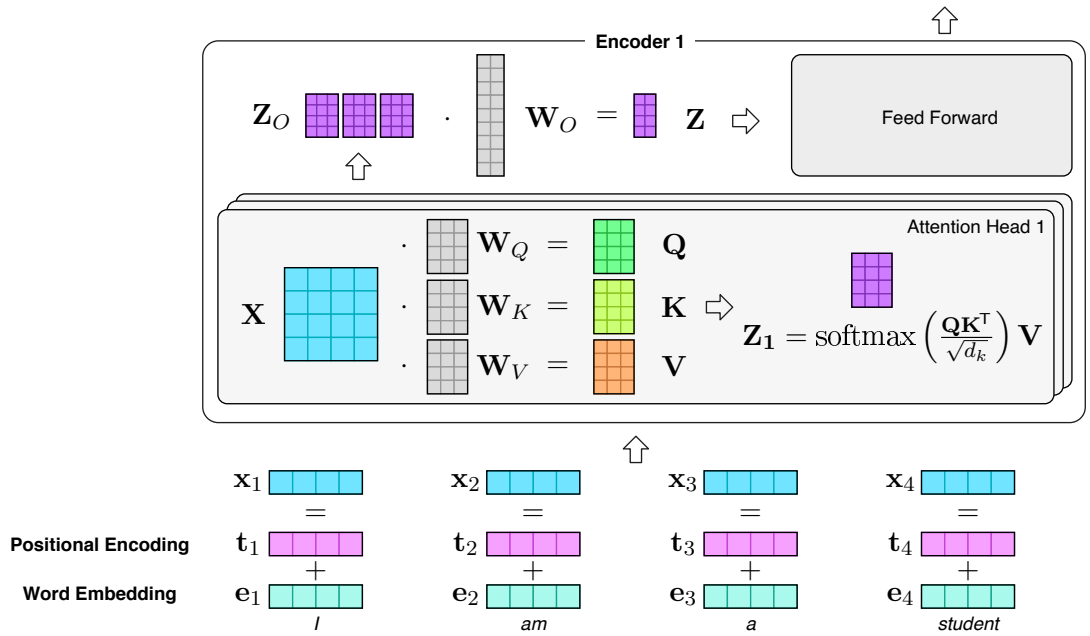


FIGURE 2.6: Simplified transformer encoder block highlighting linear projections, multi-head self-attention (matrix multiplications plus softmax), and feed-forward network.

complexity). As illustrated in Fig. 2.6, a canonical encoder layer comprises linear projections to Queries, Keys and Values, attention score computation ($\mathbf{QK}^\intercal$), scaling and softmax normalisation, weighted aggregation with $\mathbf{V}$, an output projection, followed by a position-wise feed-forward MLP; residual connections and layer normalisation wrap attention and MLP blocks.

Given an input token embedding matrix $\mathbf{X}$, the projections are:

$$\mathbf{Q} = \mathbf{X}\mathbf{W}_Q, \quad \mathbf{K} = \mathbf{X}\mathbf{W}_K, \quad \mathbf{V} = \mathbf{X}\mathbf{W}_V,$$

and the scaled dot-product attention for one head is

$$\mathbf{Z} = \text{softmax}\left(\frac{\mathbf{QK}^\intercal}{\sqrt{d_k}}\right)\mathbf{V}.$$

Multi-head attention replicates this with distinct parameter sets, concatenates head outputs and applies a final linear transformation. The dominant costs are GEMMs and softmax; intermediate attention score tensors inflate memory traffic. Long sequences or many heads stress on-chip buffer capacity and motivate tiling across batch, sequence, head and feature dimensions, as well as kernel fusion (projection + bias + activation)

and low-rank or sparse attention variants to reduce bandwidth and quadratic scaling. Vision transformers [55, 235] map image patches to token embeddings, reusing the same computational core.

Graph Neural Networks (GNNs) GNNs are a family of algorithms designed to apply machine learning techniques to relational data, falling under the broader category of geometric deep learning algorithms. The GNN model is structured in layers, where each layer's input is the node representation produced by the preceding layer. For the first layer, the node representation is provided by a feature vector associated with each node. Some of the most commonly used GNN models include the Graph Isomorphism Network (GIN) [86], Graph Attention Network (GAT) [218], and Graph Convolutional Network (GCN) [119].

The processing of each layer, as depicted in Fig. 2.7, is divided into two phases. The first phase, known as the aggregation phase, involves a sparse-dense matrix multiplication (SpMM) between the graph adjacency matrix A, where $a_{i,j} = 1$ if nodes i and j are connected, and the set of node features from the previous iteration, $H^{(k-1)}$. The graph adjacency matrix is typically stored in Compressed Row Storage (CRS) format to optimize space and computation efficiency [70, 72, 233]. The subsequent phase, referred to as the combination phase, consists of a dense-dense matrix multiplication (GEMM) between the intermediate matrix and a weight matrix W. While matrix multiplication is one common method for processing GNNs, another approach is message passing, where nodes communicate with their neighbors to update their representations.

Overall, the dimensions involved in the computations are: the number of nodes V, the number of node input features F, the number of output node features G, and the degree (number of neighbors) of each node N_v. Unlike CNNs, GNN computation is memory and latency bound due to irregular sparsity, load imbalance (varying node degrees), low arithmetic intensity during aggregation, and limited spatial/temporal locality. Efficient execution depends on sparse format selection (e.g. CRS with auxiliary indices), load balancing, and potential reordering for locality.

Embedding and Recommendation Models Large-scale recommendation models are dominated by sparse embedding table lookups (high-latency random memory accesses) followed by relatively small MLP "interaction" towers. Here, performance and energy hinge on memory capacity, bandwidth and caching of frequently accessed embedding rows rather than raw MAC throughput. Architectural features such as large, set-associative on-chip caches or near-memory compute become salient DSE variables.

Generative and Iterative Models Diffusion models and autoregressive decoders repeatedly invoke sequences of convolution, attention and MLP blocks across multiple denoising or generation steps. This temporal repetition amplifies the benefits of (i)

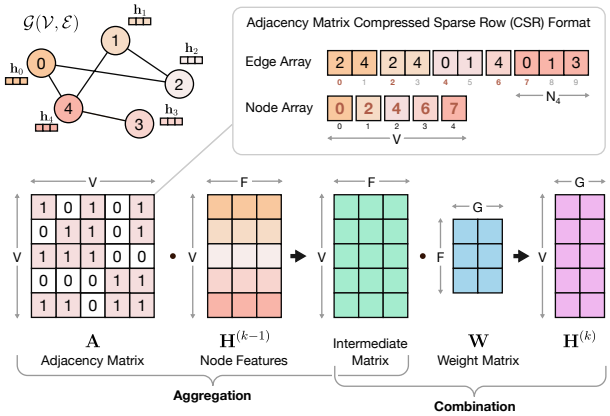


FIGURE 2.7: Abstract GNN layer: sparse neighbour aggregation (SpMM) followed by dense transformation (GEMM) and activation.

weight residency on chip, (ii) operator fusion to reduce intermediate writes, and (iii) scheduling that overlaps random number generation or conditioning data preparation with core tensor kernels.

Implications for DSE Despite architectural diversity, most workloads reduce to a small kernel set: dense and sparse matrix multiplications, convolutions, attention (a structured sequence of GEMMs plus softmax), elementwise transforms, reductions, and memory-bound embedding accesses. The salient differentiators for accelerator design are tensor shape regimes (e.g. tall–skinny vs. square GEMMs, small vs. large batch), reuse distances (spatial overlap in convolution, channel reuse in depthwise variants, head-wise reuse in attention), sparsity and irregularity (GNNs, pruned networks), and precision requirements (mixed-precision attention softmax vs. aggressively quantised MLPs). Design knobs—array topology and aspect ratio, dataflow (weight-/output-/row-stationary or attention-specialised), on-chip memory partitioning (weights vs. activations vs. temporaries like attention logits), sparsity compression formats, and adaptive precision—must be explored jointly against realistic operator mixes rather than isolated peak MAC utilisation metrics. A balanced DSA avoids over-specialisation (e.g. only large square GEMMs) that would underutilise resources on long-sequence transformers or small hidden-state RNNs, while still exploiting the high regularity of CNN and MLP blocks.

Modern AI inference workloads are compositions of a concise set of computational motifs, instantiated with widely varying structural and statistical properties. Understanding how each class stresses compute arrays, memory hierarchy, interconnect, and sparsity handling logic is prerequisite to principled design space exploration. The following sections leverage this characterisation to parameterise workloads and systematically

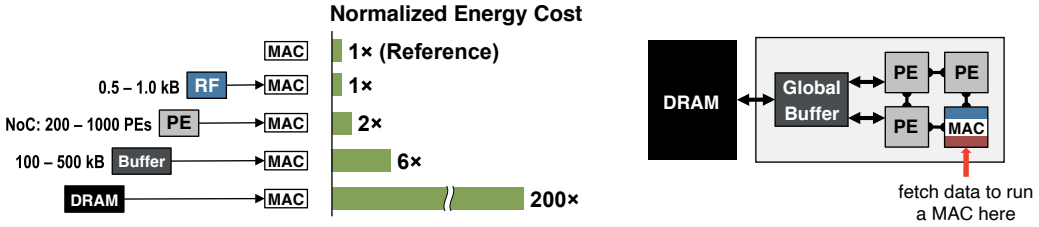


FIGURE 2.8: Normalized energy access costs in DSAs memory hierarchy [35].

evaluate architectural design points.

2.2 Domain-specific Accelerators

2.2.1 Design Objectives

When designing specialised hardware, the primary metrics considered are energy consumption and performance. However, for a specific application, additional metrics may also be relevant [212]. For instance, accuracy is crucial when safety or quality are at stake; throughput is a typical performance measure, but latency is particularly important for real-time applications. Power consumption impacts heat dissipation, while total energy is critical for embedded systems with limited power budgets. Fabrication cost, determined by silicon area and memory bandwidth requirements, can also be significant. Scalability and flexibility enable a single design to address a wide range of scenarios, from IoT devices to data centres, thereby reducing design costs. These metrics often trade off against one another, so designers must balance them according to the task.

Energy efficiency in modern computing systems is increasingly limited by memory access costs. As shown in Fig. 2.8, accessing off-chip DRAM consumes much more energy than an ALU operation. This is further confirmed by Fig. 2.9, which highlights that the energy gap between memory and computation persists across technology generations. Both general-purpose processors and DNN accelerators feature a memory hierarchy, but while CPUs use caches, accelerators often employ explicitly managed scratchpad buffers, as data movement patterns are predictable at compile time. Accessing larger memories is inherently more expensive, so minimising data movement, especially to DRAM, is key to energy efficiency. Reducing operand precision can also save energy at the expense of some accuracy [211].

Throughput and latency are improved primarily via parallelisation, which depends on the number of Processing Elements (PEs) and their associated MAC units. Efficient data movement remains critical: memory bandwidth bottlenecks can cause idle cycles and increase latency [211].

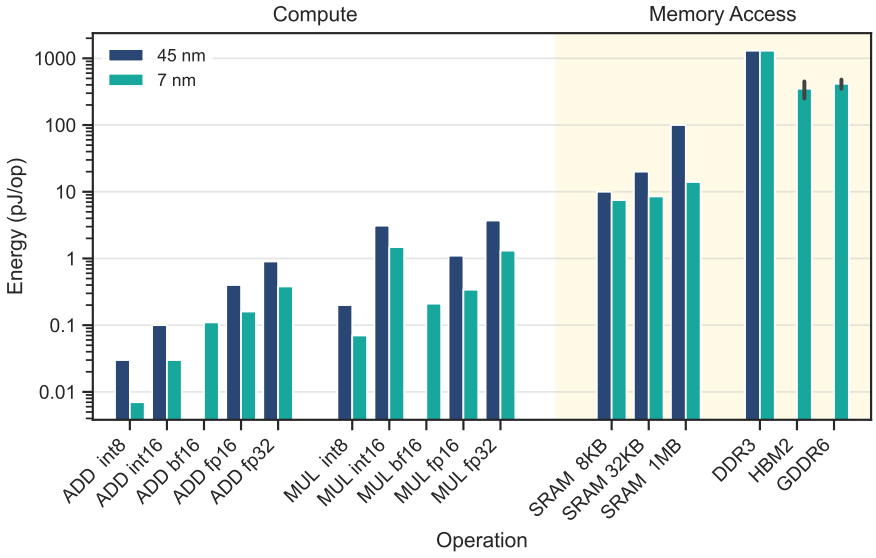


FIGURE 2.9: Energy-per-operation comparison between compute and memory and different technology nodes [97, 107].

2.2.2 Spatial Architectures

CPUs and GPUs execute deep learning tasks using parallelisation techniques such as SIMD, and are classified as temporal architectures because ALUs interact only through memory. In contrast, most DNN DSAs use spatial architectures, deploying many PEs to exploit DNN parallelism (see Fig. 2.8). Each PE typically has local scratchpad memory and a MAC unit, while a shared global buffer supplies data. The PEs and global buffer are interconnected via a Network-on-Chip (NoC), whose topology and flexibility depend on design choices. Spatial architectures enable direct communication between PEs, increasing data reuse and reducing memory accesses [211]. Various *dataflows*, based on data movement patterns, can be employed, as discussed below.

2.2.3 Data Reuse

Data movement can be reduced by exploiting the fact that DNNs often reuse the same data across multiple MAC operations, a property known as **data reuse**. Inputs, weights, and outputs each exhibit distinct reuse patterns. Kwon et al. [126] identify two key behaviours enabling reuse: **multicast** and **reduction**. The former refers to sharing the same data point across space (multiple PEs in parallel) or time (repeated use in subsequent cycles), reducing the need for repeated memory accesses. The latter pertains

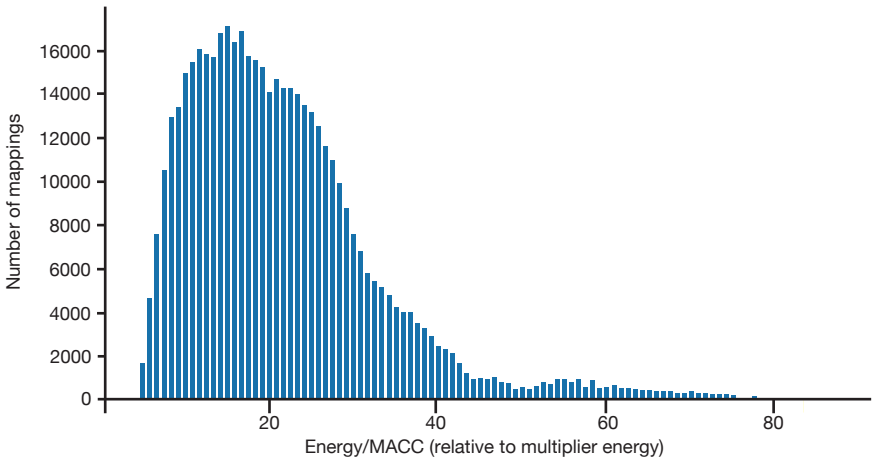


FIGURE 2.10: Histogram of the energy efficiency of various mappings of a VGG convolutional layer on an example architecture [164].

to accumulating partial sums, either spatially (forwarded to adjacent PEs) or temporally (retained in a local buffer).

For convolutional layers, Chen et al. [35] distinguish three types of reuse: (i) **filter reuse**: each filter is reused across all inputs in a batch. (ii) **input feature maps reuse**: each input is reused by all output channels. (iii) **convolutional reuse**: overlapping filter windows cause further reuse of both filters and inputs, depending on stride. While accesses for each data type (inputs, weights, outputs) can be minimised individually, it is generally impossible to do so for all simultaneously, as reusing one typically precludes reuse of another in the same operation. Thus, data movement optimisation relies on **mapping** and scheduling MAC operations both in time and space. DSAs may support multiple mappings and dataflows, or only a subset depending on hardware flexibility [211].

2.2.4 Mapping

Optimal mapping minimises energy and other costs by specifying:

1. The temporal and spatial execution order of MAC operations.
2. How data is tiled and moved across the memory hierarchy and NoC.

Numerous potential mappings exist for a DNN layer, with the optimal mapping contingent on the capabilities of the hardware and the selected optimization criterion. Fig. 2.10 underscores the significance of optimizing the mapping by illustrating the 19× variability in energy efficiency across a range of potential mappings.

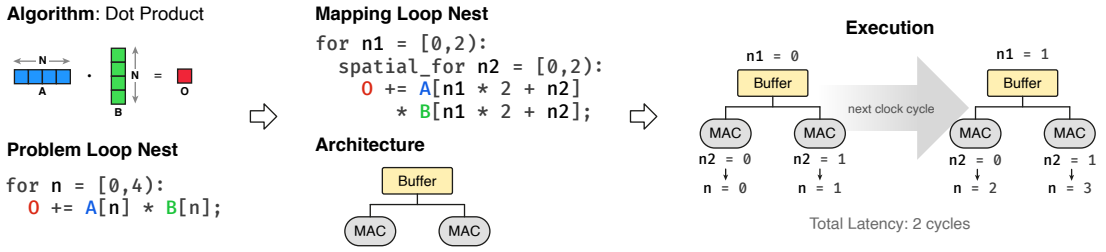


FIGURE 2.11: Loop Nest Representation of a Dot Product operation and its mapping on a simple architecture

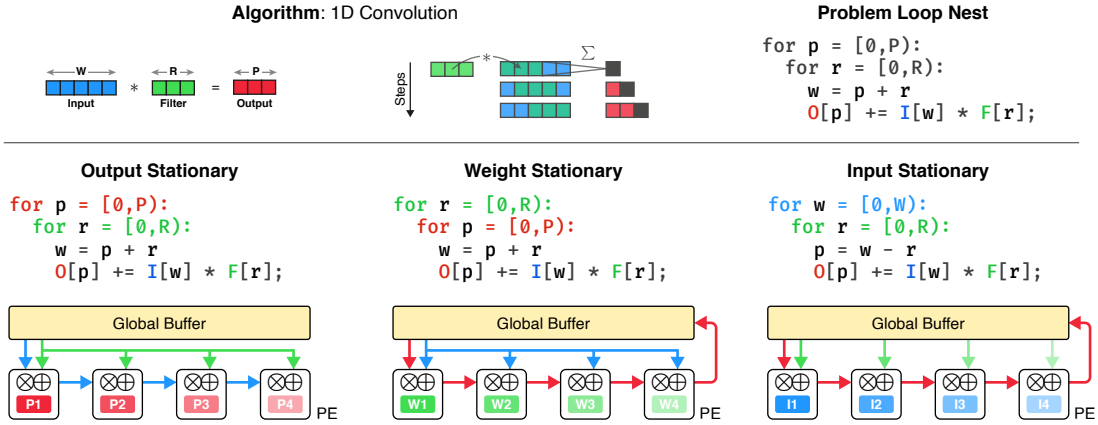


FIGURE 2.12: 1D Convolution and Mapping Dataflows

Tensor operations are often expressed as loop nests (see Fig. 2.11). Mapping these loops to hardware involves four main choices at each memory or compute level:

- **Loop Permutation (Ordering)**: Determines which loops are outermost or innermost, affecting data stationarity and memory access patterns.
- **Loop Tiling (Index Factorisation)**: Divides loops into tiles to fit data into buffers.
- **Loop Unrolling (Spatial Mapping)**: Assigns parallel workload portions to different hardware elements, inducing specific communication patterns.
- **Datatype Bypass**: Decides which data types are stored or bypassed at each buffer level.

Dataflow	N. of MACs	Output		Weight	Input
		Reads	Writes	Reads	Reads
Output Stationary	$P \cdot R$	**0**	**P**	$P \cdot R$	$P \cdot R$
Weight Stationary	$P \cdot R$	$P \cdot R$	$P \cdot R$	**R**	$P \cdot R$
Input Stationary	$P \cdot R$	$P \cdot R$	$P \cdot R$	$P \cdot R$	**P**

TABLE 2.1: Dataflow buffer accesses comparison [211]

2.2.4.1 Loop Permutations and Dataflows

Dataflow determines which data types remain stationary in low-level buffers, and which are fetched from higher-level memory, depending on workload shape and tensor sizes. Chen et al. [35] classify dataflows by the stationary data type. Fig. 2.12 shows three common dataflows for 1D convolution:

- **Output Stationary**: The output index changes least often, so partial sums are accumulated locally until the output is complete. This allows spatial multicast of weights and forwarding of inputs.
- **Weight Stationary**: By changing loop order, weights become stationary; other data is fetched as needed. Inputs can be multicasted, and partial sums forwarded for reduction.
- **Input Stationary**: Inputs remain stationary, weights are unicasted, and partial sums are reduced spatially.

Table 2.1 compares buffer access costs for each dataflow. The optimal dataflow depends on workload size and shape, and must be supported by the hardware.

2.2.4.2 Loop Tiling and Buffering

Loop tiling (index factorisation) divides loops to fit data into different buffer levels, as illustrated in Fig. 2.13 for a 2-level memory hierarchy. The tile size for each data type at buffer level L is:

$$\text{tile size}^{L}_{\text{data type}} = \prod_{d=1}^{D} \prod_{l=1}^{L} f_{d,l} \cdot r_d \tag{2.1}$$

where $f_{d,l}$ is the loop bound for dimension d at level l, and r_d indicates relevance of the dimension. Buffers have limited capacity, so the mapping must fit within buffer constraints. Datatype bypass allows, for example, weights to be fetched from higher-level buffers to reduce lower-level buffer requirements.

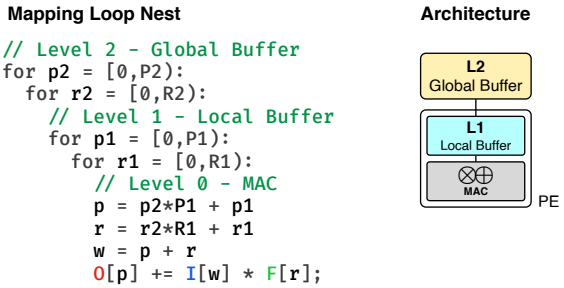


FIGURE 2.13: Temporal Tiling in 1D Convolution mapping on a 2-level memory hierarchy architecture

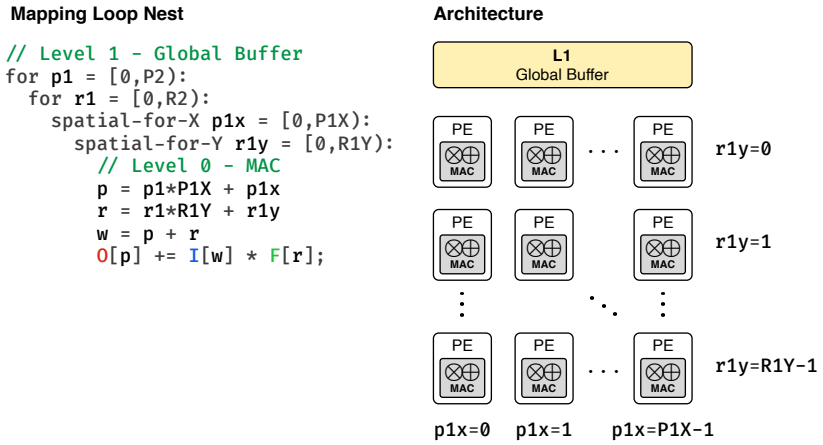


FIGURE 2.14: 1D Convolution Spatial Mapping

2.2.4.3 Loop Unrolling and Spatial Mapping

Spatial mapping exploits parallelism by unrolling loops across multiple hardware instances (PEs), as represented by `spatial-for` directives. If the loop dimension exceeds the number of spatial units, tiling is also needed. In 2D PE meshes, spatial mapping can be applied along horizontal or vertical dimensions, generating different communication topologies (unicast, multicast, broadcast) based on the data type. Fig. 2.14 demonstrates spatial mapping for 1D convolution.

2.2.4.4 Advanced Loop Nests

While previous examples used 1D convolution, DNN workloads generally involve matrix multiplications and 4D convolutions. Fig. 2.15 shows the loop nest for a convolutional layer. The same mapping principles—loop tiling, permutation, and spatial mapping—apply, but with higher dimensionality and complexity. Fig. 2.16 provides

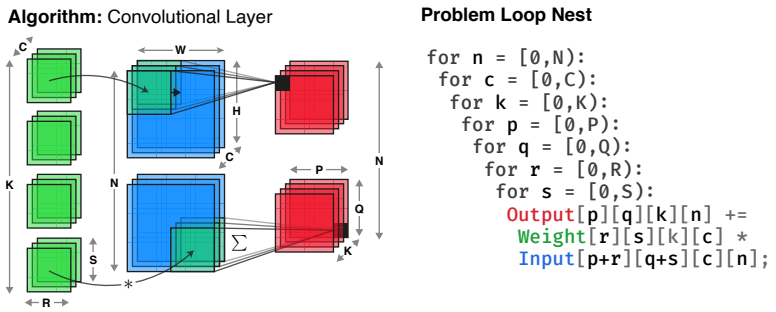


FIGURE 2.15: Convolutional Layer loop nest representation

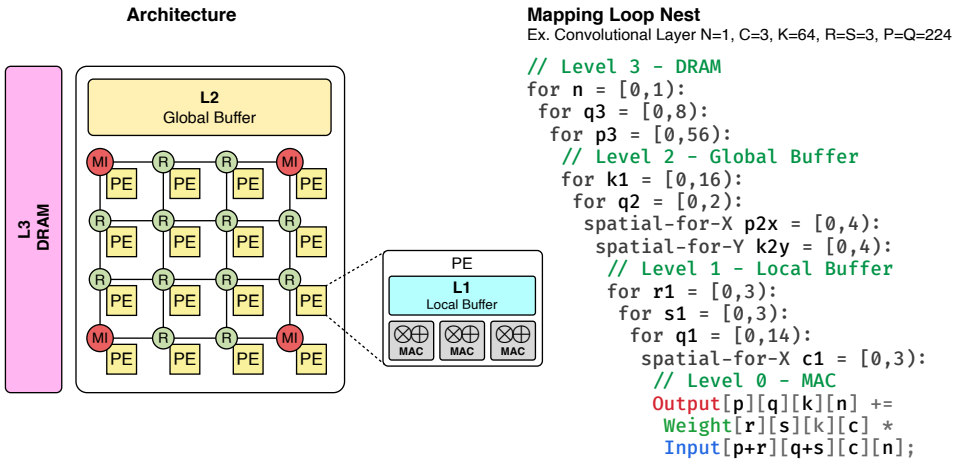


FIGURE 2.16: Convolutional layer mapping loop nest example

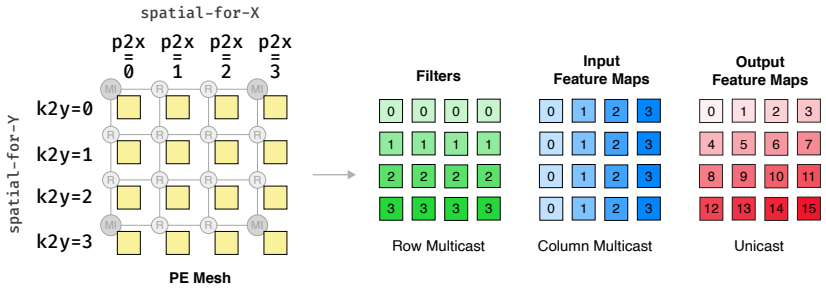


FIGURE 2.17: Communication topologies in NoC arise from spatial mappings

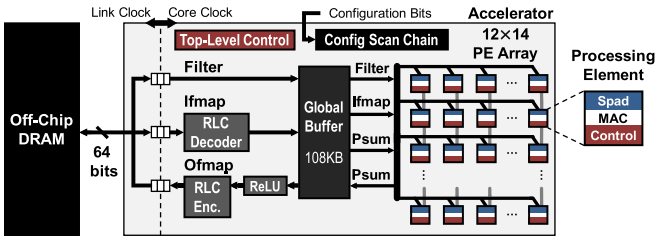


FIGURE 2.18: Eyeriss system architecture [35]

an example mapping for a convolutional layer on an architecture with a 3-level memory hierarchy and a 4×4 PE mesh, each PE containing a local buffer and multiple MAC units. The number of weights stored at each buffer level can be computed via Eq. (2.1). The choice of spatial mapping indices determines multicast opportunities in the NoC; for example, applying `spatial-for-X` to a dimension relevant to weights enables their reuse across rows, as illustrated in Fig. 2.17.

2.2.5 Published designs

Now, we report some accelerator designs published in the literature that constitute the state of the art. Some of these implement the previously described dataflows, others implement new ones, and others can support several.

Eyeriss Eyeriss [35] is a deep CNN accelerator. The test chip was implemented in 65nm CMOS features a spatial array of 14x12 PEs fed by a reconfigurable multicast NoC that handles many layer shapes and minimises data movement by exploiting data reuse, a 108KB global buffer, ReLU and feature map compression units, as shown in Fig. 2.18. The chip communicates with the off-chip DRAM using a 64-bit bidirectional data bus. The image data and filter weights are read from DRAM into the buffer and streamed into the spatial computation array, allowing for overlap of the memory traffic and computation (**double buffering**). One of the main contributions of Eyeriss is the introduction of the **Row Stationary** dataflow, which differently from the dataflows described in Section 2.2.4.1, aims to maximise reuse and accumulation at the lowest memory level (Register Files in the proposed architecture) for all data types at the same time. The Row Stationary dataflow first divides the 4D convolution into 1D convolution primitives that can all run in parallel. Each primitive operates on one row of filter weights and one row of Input Feature Maps, and generates one row of Partial Sums. The Partial Sums from different primitives are further accumulated together to generate the Output Feature Maps rows. The 2D array of PEs and the row-stationary dataflow allow different forms of data reuse that reduce accesses to the Global Buffer. For example, each filter row is reused across multiple PEs horizontally. Each row of input activations is reused

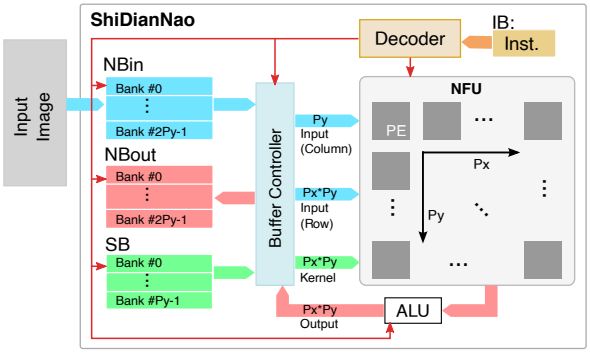


FIGURE 2.19: ShiDianNao system architecture [57]

diagonally across multiple PEs. Each row of partial sums is further accumulated across the PEs vertically [211].

ShiDianNao ShiDianNao [57] is a CNN accelerator that targets visual recognition applications. The architecture is shown in Fig. 2.19 and consists of the following main components: two buffers for input and output activations, named Neuron Buffer Input (NBin) and Output (NBout) respectively; a buffer for weights, named Synapses Buffer (SB); a neural functional unit (NFU) and an arithmetic unit (ALU) for activation function calculations. All computations are performed with 16-bit fixed-point operands. This results in negligible accuracy loss with respect to traditional 32-bit floating-point precision, but significantly reduces hardware costs. The NFU of ShiDianNao is where the convolution computation takes place and consists of a 2D mesh of PEs. Output-stationary is the imlemented dataflow; thus, each PE handles the processing for each output activation value and input activations are fetched from neighbouring PEs, through queues and a specialised NoC. As required by the output-stationary dataflow, the partial sums are accumulated locally in each PE and the weights are spatially multicast from the Neuron Buffer to the PEs.

Simba Simba [200] is a Multi-Chip-Module (MCM) based scalable multi-accelerator system that consists of homogeneous sub-accelerators (SAs) integrated at package level and connected by a Network-on-Package (NoP). MCM packaging approaches can reduce cost by employing smaller chiplets connected together post-fabrication, as yield losses cause fabrication cost to grow super-linearly with die size. Furthermore, thanks to high-speed package-level signaling, an appropriate number of chiplets can be integrated based on the target application, enabling high scalability. Fig. 2.20 shows the Simba architecture at three different levels: package, chiplet, and PE. The Simba prototype consists of a 6x6 array of chiplets interconnected via a mesh NoP. Each chiplet in Simba is a

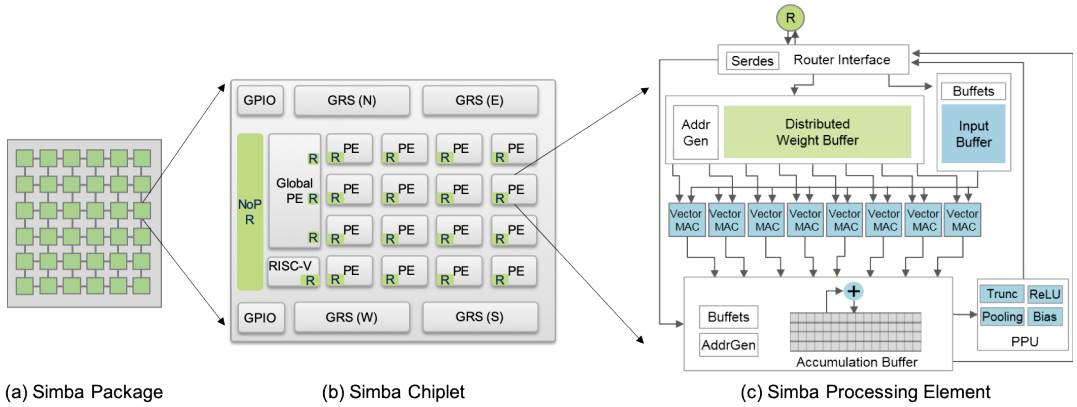


FIGURE 2.20: Simba architecture from package to PE [200]

standalone sub-accelerator and consists of a Global Buffer and a 4x4 array of PEs interconnected through a NoC, a RISC-V controller, a NoP router, and 4 ground-referenced signaling (GRS) devices for high-speed intra-chiplet communication. Each PE is similar to the NVIDIA Deep Learning open accelerator design (NVDLA) [157] and consists of a weight buffer, an input buffer, an array of vector MAC units, an accumulation buffer for partial sums, and a post-processing unit (PPU) for pooling and activation function operations. Each vector MAC unit performs a dot product 8:1 across the dimension C of the convolution workload (input channels). The Simba PE adopts the weight-stationary dataflow; thus, weights remain in the vector MAC registers and are reused across iterations, while new inputs are read every cycle.

A larger number of chiplets can be useful to increase throughput in data centre scenarios. Furthermore, Simba allows cross-layer pipelining across multiple chiplets (allowing concurrent processing of different layers on multiple chiplets) and non-uniform tiling strategies (for inter-chiplet communication latencies compensation).

2.2.5.1 Flexible Dataflow Accelerators

The DNN accelerators just recalled support only fixed dataflow patterns internally. Many accelerators are only optimised for traffic within a convolutional layer and their interconnection networks only support a limited set of communication patterns. This makes it challenging to map arbitrary dataflows on the hardware efficiently and can lead to underutilisation of the available compute resources. DNN accelerators need to be programmable to enable maximum flexibility. For them to be programmable, they need to be configurable internally to support the various dataflow patterns that could be mapped over them. Some works in literature explore supporting multiple dataflows in a single flexible hardware, including EyerissV2 [36], FlexFlow [141], and MAERI [129].

The main features of these designs are flexible NoCs and the support of various memory access patterns that allow to execute different dataflows that have different numbers of loop levels and loop ordering in the loop nest.

Heterogeneous Dataflow Accelerators The dataflow flexibility comes at the cost at the cost of extra hardware components (switches and wires) and energy consumption. Furthermore, reconfiguring for the optimal mapping for each layer can also cause latency overhead. Kwon et al. [128] proposed a new class of DNN accelerators called heterogeneous dataflow accelerators (HDAs). These accelerators combine the multi-accelerator structure of Simba with the possibility to leverage multiple dataflows. This goal is achieved by integrating multiple heterogeneous sub-accelerators, each deploying a different dataflow. Therefore, each layer of a DNN model can be assigned to the SA that is capable of running it efficiently without having to reconfigure dataflows. This capability is useful in multi-DNN scenarios, i.e., applications in which multiple DNN models are involved for several sub-tasks. This is the case of Virtual Reality (VR) and Augmented Reality (AR) applications in which several models, including models for object-recognition and depth estimation, are applied jointly. The diversity in shapes and sizes of layer of different models introduce challenges for DNN accelerator with fixed dataflows that can be overcome with HDA.

Sparse Accelerators An interesting property of some DNN models is sparsity. Data are sparse when there are many repeating values, in most cases repeating zeros. In these cases, sparsity measures the fraction of data that is zero. Sparsity can be leveraged to reduce data access costs because sparse data can be compressed. Furthermore, it allows to reduce MAC operations because of the algebraic zero-product property. In particular, if one of the operands is zero, the MAC operation can be skipped to save energy and execution cycles. In DNN sparsity can arise in both weights and input features maps. Sparsity in weights comes from the pruning operation in which some connections between neurons are removed (the corresponding weight is set to zero) while trying to maintain the original accuracy. Sparsity in input feature maps comes from activation functions such as ReLU. Several compression techniques have been elaborated to efficiently store sparse tensors in buffers and many DSAs have been designed to take advantage of sparsity in both inputs and weights tensors [211].

For example, in the Eyeriss [35, 36] design, energy consumption is reduced both by avoiding reading operand values and running the multiplier when an activation is zero as this saves accesses to input operands, writes/updates to output operands, and arithmetic computation. The SCNN [165] (Sparse CNN) accelerator architecture, instead, improves performance and energy efficiency by exploiting both zero-valued weights and zero-valued activations. Specifically, SCNN employs a novel dataflow that enables the sparse weights

and activations to be maintained in a compressed encoding, which eliminates unnecessary data transfers and reduces storage requirements.

GNN Accelerators Graph data is irregular when compared to image, audio and tabular sources. This motivated the rise of specific GNN accelerators [195, 233] featuring either fixed dataflows or adaptive capabilities, which outperform conventional GPU and CPU execution enabling high performance thanks to parallelization and energy efficiency due to high specialization. Differently from traditional DNN workloads such as fully connected and convolutional layers [211], a layer of a GNN is organized in two phases and an *intra-phase dataflow* must be selected for each of them. Aggregation and combination are implemented as two loop nests executed sequentially. The matrices involved are the same as in Fig. 2.7. In each phase, the intra-phase dataflow is specified by the loop ordering and the spatial loop tiling and unrolling. The global buffer and the PEs are interconnected through a NoC for data movement. According to the unrolling dimensions, data movement between PEs may be needed for spatial reduction, i.e., accumulation of partial sums, or multicast capabilities could be useful to reduce memory accesses. Generally, each dataflow choice requires specific microarchitectural implementations. Furthermore, the dataflows of the two phases are interdependent. The interaction between the aggregation and combination phases is described by the *inter-phase dataflow* and determines the amount of memory accesses needed to move data from one phase to the other.

2.3 Design Space Exploration

Accelerator architectures for DNNs are highly specialised and often specific to a particular task. For DSAs to be easily used in application scenarios, it is necessary to quickly explore all possible design choices. The traditional implementation in Register Transfer Level (RTL) to study the effects of a design choice is too low-level and time consuming. In order to speed up the evaluation flow, it is necessary to develop and use simulation tools, to obtain estimates on the energy consumption and performance of a design, and tools for DSE for the rapid search for appropriate design solutions for the task under consideration. As we discussed, the number of accesses at each level of the memory hierarchy, the traffic in interconnection networks, and the degree of data reuse depend on how the fundamental operations of a DNN workload are mapped temporally and spatially on the accelerator hardware. The energy and latency required to execute a layer of a DNN also depend on the mapping. For this reason, several analytical models for mapping evaluations and architectural simulations have been proposed in the literature, such as Timeloop/Accelergy [164, 229], ZigZag [144] and MAESTRO [127]. These simulation models are often used by algorithms for the exploration of the mapping space which in the case of flexible dataflows cannot be explored exhaustively

[89, 99, 190]. Given the large amount of possible architectures and mappings, it is clear that many aspects must be considered during the design of an accelerator, such as the number of spatial computing and buffer instances, memory hierarchy, buffer sizes, spatial and temporal mapping, and data allocation. Exploring exhaustively the whole space of possible designs is an NP-hard problem and can be very costly, especially if the target platform of the accelerator is an ASIC. For this reason, there is growing interest in the development of tools and frameworks for efficient design space exploration [29, 73, 110]. DNN specific accelerators also offer the opportunity to implement hardware approximation [7, 191], scalable-precision MAC computing [102] and data compression [52, 192] techniques. These allow to further reduce energy consumption and, in the case of data compression, to increase throughput in non-compute-bound situations, at the cost of the model accuracy reduction. Some DNN layers are more sensitive to approximations than others [206]. Therefore, these techniques should be applied accurately by the exploration framework that trades off energy reduction and accuracy loss [65].

In Section 2.3.1 we introduce some state-of-the-art simulation models. Then, in Section 2.3.2, we report some tools that search for the best mapping for a single layer, called **mappers**. In Section 2.3.3, we introduce tools for hardware mapping **co-design** of DNN accelerators capable of simultaneously searching for the best combination of mapping and hardware architecture. Finally, in Section 2.3.4 we introduce **global schedulers** for multi-DNN applications.

2.3.1 Simulation Models

Architectural simulation models are vital in computer architecture research, enabling fast and accurate design space exploration and early evaluation of architectural choices. They are widely used for CPU development [21], and more recently, to assess DNN inference accelerators by quickly providing metrics for varied architectures and mappings.

MAESTRO MAESTRO [127] is a DNN accelerator cost model that uses data-centric directives to concisely specify DNN dataflows, distinct from the traditional loop nest approach. MAESTRO (Modelling Accelerator Efficiency via Spatio-Temporal Reuse and Occupancy) estimates metrics such as execution time and energy efficiency for a given DNN and hardware configuration. It takes as input the DNN model, dataflow description, and hardware setup, and outputs estimates for execution time, component-wise energy, and NoC costs. According to the authors, MAESTRO achieves 90–95% accuracy compared to open-source RTL, while being 1,000–4,000× faster. Its hardware model assumes a 2-level memory hierarchy and does not support datatype bypass mappings.

STONNE STONNE (Simulation TOol of Neural Network Engines) [150] is a cycle-accurate, modular, and extensible open-source framework for evaluating flexible dataflow

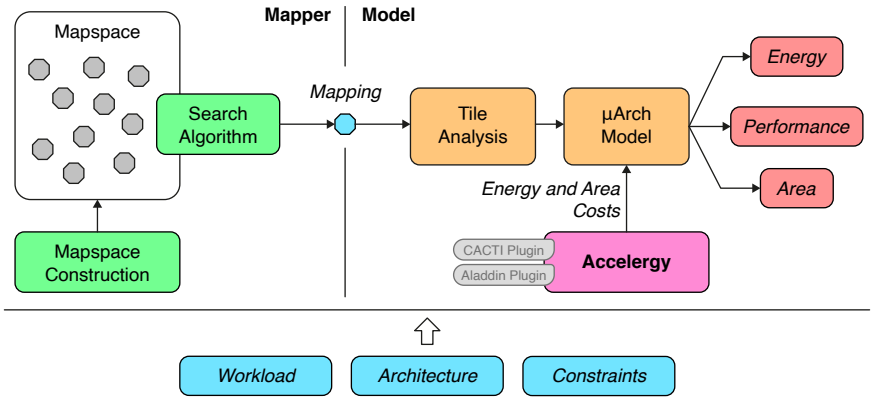


FIGURE 2.21: Timeloop/Accelergy framework overview

accelerator architectures on full DNN models. The model for the MAERI architecture [129] has been validated against BlueSpec Verilog results. Implemented in C++, STONNE takes DNN models in Caffe DL format. While it delivers accurate, cycle-level simulation, it is slower than analytical models.

Timeloop Timeloop [164] is an analytical tool for rapid evaluation of DNN accelerator mappings and hardware architectures. It features a *mapper* to construct and explore the mapspace of possible mappings, and a *model* to estimate energy, area, and performance for each mapping (see Fig. 2.21).

Inputs for Timeloop are:

- **Workload**: Supported workloads are expressible as deep loop nests with fixed bounds and linear operand indexing, typically corresponding to DNN layers. Each MAC operation is a point in the operation space, with projections onto input, weight, and output dataspaces described in the workload file.
- **Architecture**: Hardware organization is described as a hierarchical storage tree with arithmetic units (e.g., MACs) at the leaves and DRAM at the root. Buffer capacities and micro-architectural parameters, as well as the number of spatial instances, are specified.
- **Mapping Constraints**: Constraints allow fixing aspects such as loop ordering and spatial bypass, reflecting architectural limitations or exploration choices.
- **Mapspace Exploration**: The mapper supports exhaustive and random search algorithms, configurable by search parameters and stopping criteria. Mappings are evaluated against constraints and buffer capacities.

Alternatively, users can evaluate a specific mapping by providing the corresponding input files, bypassing mapspace construction. Timeloop's analytical model evaluates action counts for architectural components (e.g., buffer reads/writes, network accesses) using tile analysis, which estimates data movement between buffers without full loop execution. This yields energy and performance insights, especially when combined with Accelergy [229] for detailed energy and area estimation. Timeloop has been extended to support sparse workloads (Sparseloop [230]) and improved NoC evaluation with LAMBDA [189].

OMEGA OMEGA [70] is a cycle-accurate GNN dataflow simulation framework built upon STONNE [150]. The simulated architecture consist in a combination of MAERI [129] flexible-dataflow accelerator for dense computations and SIGMA [176] for sparse matrix multiplications (SpMM). The results from the two simulated architectures obtained through STONNE are combined in analytical model to extract relevant metrics regarding the GNN inter-phase dataflows.

2.3.2 Layer Mappers

As already reported, state-of-the-art DNN accelerators typically consist of large arrays of processing elements for parallelism paired with a multi-level memory hierarchy and a flexible NoC to improve data reuse. These architectural structures improve the performance and energy efficiency of DNN execution, but they increase the number of possible scheduling parameters that the designer has to select. The spatial and temporal scheduling in DNN accelerator is called mapping and can be using the loop nest notation. The combination of these parameters is the mapping and describes how a DNN layer is scheduled spatially and temporally on an accelerator. Taking into account a target DNN layer and a specific hardware architecture, there could be millions, or even billions, of valid mappings with a wide range of performance and energy efficiency [164]. Given the high variability in DNN layer shapes and dimensions and in accelerator hardware architectures, it is not trivial to manually select the appropriate mapping parameters. For this reason, many **mappers**, i.e., automatic scheduling frameworks for DNN layers, have been proposed in the literature. Some of them, based on analytical cost models, prune the mapspace based on hardware constraints and then perform a slow exhaustive search for the best mapping for an objective metric. Exploring the mapspace in such a brute-force fashion can easily become intractable for larger DNN layers and more complex hardware architectures [99]. Other approaches considered are feedback-driven, such as black-box optimisation and other machine learning algorithms, or heuristic-based. Some effort was also employed to formulate the mapping problem as a linear optimisation problem.

Timeloop Mapper As already mentioned, Timeloop [164] comes with its own mapper module. The search algorithms implemented are basically random and exhaustive. In absence of sufficient mapping constraints, so in totally flexible hardware situations, the Timeloop Mapper struggle to find good mappings.

Recently, Timeloop was extended with a new mapspace constructor called Ruby [96], which introduces the possibility to consider imperfectly factorised tensor dimensions for loop bounds. This means that loops can have remainders and, despite increasing the mapspace size, it allows to find mappings with 20-50% energy-delay product improvements with respect to traditional mappings. The imperfect factorisation is particularly useful to perform a spatial mapping that increases the hardware utilisation.

CoSA CoSA [99] is a constrained-optimisation-based approach for scheduling DNN accelerators. Differently from other mappers that require either exhaustive brute-force-based or expensive feedback-driven approaches, CoSA expresses the DNN accelerator mapping problem as a constrained-optimisation problem that can be deterministically solved using mathematical optimisation libraries in one pass. Leveraging the regularities in both DNN layers and spatial hardware accelerators, CoSA formulates the DNN scheduling problem as a prime factor allocation problem that determines 1) tiling sizes for different memory levels, 2) relative loop ordering to exploit reuse, and 3) how computation should be executed spatially and temporally. CoSA constructs the scheduling constraints by exposing both the algorithmic behaviours (e.g., layer dimensions) and hardware parameters (e.g., memory and network hierarchies). Together with clearly defined and composable objective functions, CoSA can solve the DNN scheduling problem in one shot without an expensive iterative search. CoSA makes use of the Timeloop analytical simulation model and the authors demonstrate that CoSA can generate mappings that outperform the Timeloop mapper by 2.5× across different DNN network layers, while requiring 90× less search time. Defining the objective function is a non-trivial task as it should strongly depend on the target hardware architecture. Furthermore, it is important to consider the bandwidth of the buffers in the architectural hierarchy.

GAMMA GAMMA (Genetic Algorithm-based Mapper for ML Accelerators) [109] is a domain-specific genetic algorithm-based method for DNN accelerator mapping. GAMMA performs a complete search, considering three mapping choices (loop tiling, loop permutation, and spatial mapping strategy). The key novelties in GAMMA are: a specialised genetic encoding of all three aspects of DNN mapping, specialised mutation and crossover operators to evolve new mappings, and new genetic operators to model the behaviour of adding and removing levels of parallelism. GAMMA is part of a closed-loop workflow that leverages the MAESTRO analytical cost model. Due to the simulation adopted, GAMMA can explore mapping for hardware architecture with only two tiling levels (Global Buffer and Local Buffer) and does not support datatype bypass.

Others Among the other mappers proposed in the literature, we mention MindMappings [89], which makes use of a surrogate reinforcement learning model to replace the Timeloop simulation to speed up the solution evaluation time. Mappings are tested on the trained surrogate model before doing heavyweight simulation, and this allows speeding up search time. The drawback of this approach is that training the surrogate model is necessary. LOMA [210] leverages loop permutation generation and analysis in order to heuristically allocate more data in lower buffers in the architectural hierarchy and minimise costly accesses to higher-level ones.

2.3.3 Co-Design Tools

Mapping and hardware resources are deeply interdependent. A mapping can be valid for a specific accelerator because the buffer capacities and spatial instances (e.g., PEs) are sufficient. Buffer capacities and number of spatial instances determine not only if a mapping is valid or not, but also if this is the best. The same mapping can generate different energy figures for two different accelerators because of different buffer sizes that cause different access energy costs. Instantiating too many spatial instances or too large buffers can cause under-utilisation and unnecessary area costs if proper mappings cannot be obtained for a specific workload. These are some of the reasons why the co-design of hardware and mapping can be useful for designing solutions that are efficient and performing while also minimising costs. Some works have been proposed in this direction, some targeting single layers on single accelerator systems and others focusing on global schedulings for multiple layers.

ZigZag ZigZag [144] is a memory-centric DNN accelerator DSE framework that allows architecture and mapping search. The main novelties are the possibility to explore different memory hierarchies considering an area budget and the introduction of the *uneven mapping* concept that extends the mapping space. Uneven mappings are expressed through a memory-centric dataflow representation based on an enhanced nested-for-loop format and consist of having a different loop allocation in buffers for each datatype. ZigZag also comes with its own analytical hardware cost model that can estimate the performance and energy metrics of an accelerator. The mapping space is explored implementing various heuristic search algorithms based on data stationarity, reuse, and early cost estimations.

Herald Kwon et al. [128] proposed the Heterogenous Dataflow Accelerator class. They consist of multiple sub-accelerators (SA) with different dataflows. In order to achieve high efficiency, it is necessary to carefully allocate hardware resources and also optimise the layer execution scheduling on the sub-accelerators. For this purpose, Kwon et al. [128] also introduce Herald, a framework for co-optimising hardware partitioning and layer scheduling. Unlike layer mappers, Herald targets layer granularity execution on

each sub-accelerator of HDAs and is capable of scheduling multi-DNN workloads. The design space exploration is guided by the MAESTRO [127] analytical cost model extended to support HDAs. Herald is also capable of optimising the hardware resource assignment to different sub-accelerators. The user defines the number of SAs and their dataflows and provides a PE budget, Herald finds the best PE assignment for each SA. For example, using Herald on a benchmark workload, the authors identify a promising HDA architecture combining a NVDLA sub-accelerator with an Eyeriss-like sub-accelerator, which demonstrates 65.3% lower latency and 5.0% lower energy compared to the best fixed dataflow accelerators.

Planaria Unlike Herald architecture, in which the PE hardware is optimised in an offline fashion, Planaria [73] architecture can dynamically partition into multiple smaller DNN engines at runtime. This microarchitectural capability allows to spatially colocate multiple DNN model on the same hardware. This flexibility is achieved with omnidirectional systolic arrays that allow flow of data in all four directions in the array. These omnidirectional systolic arrays are organised in on-chip pods, namely groups of four systolic arrays, that also include interconnection and shared storage and enable the coordinated partitioning with on-chip and off-chip data transfer. The authors also introduce a multi-tenant dynamic scheduling algorithm that leverages the dynamic hardware partitioning to significantly improve utilisation, throughput and SLA meeting.

2.3.4 Multi-Tenant and Multi-DNN Schedulers

With the growing demand for computing for DNN workloads, many large accelerators are spreading to both the edge and the cloud, for example the aforementioned scalable Simba [200] accelerator. Additionally, many designs employ multi-accelerator solutions with homogeneous [200] or heterogeneous dataflow sub-accelerators [128]. In order to maximise the utilisation of hardware resources, tools for optimising scheduling at **layer granularity** are necessary [11]. Such schedulers often operate in multi-tenant and multi-DNN, also called multi-workload, environments.

The term **multi-tenant** is often used in data center scenarios where an accelerator serves multiple users and has to satisfy the Service level agreement (SLA). This means that the accelerator receives inference requests for different DNN models and has to meet Quality of Service (QoS) constraints, such as latency. These accelerators can often run multiple DNN layers simultaneously while meeting deadlines, thanks to an optimised scheduling algorithm. These scheduling algorithms are often dynamic or online, i.e., they decide on the fly the resources to assign to the execution of a certain layer as soon as a new request arrives. The term **multi-DNN** is used more across the board from edge to cloud scenarios. It often indicates situations where multiple DNN models are employed for a given task and therefore run simultaneously in the same accelerator. An example can be an augmented reality (AR) application in which image classification,

object detection and depth estimation models are employed on the same input image. In these situations the scheduling algorithm can be dynamic but also static because the workload is defined in advance and does not vary over time.

The already introduced Herald framework [128] and Planaria [73] architecture combine a multi-DNN scheduler with a hardware partitioning algorithm. MAGMA [110] is a multi-tenant scheduling algorithm targeting independent jobs, i.e., it does not consider layer inter-dependencies, and multi-accelerator systems and it is based on a genetic algorithm without relying on heuristics. PREMA [40] is a multi-tenant dynamic scheduler that target single-accelerator but preemptible systems that optimize for maximum SLA satisfaction rate. AI-MT [11] proposes a novel homogenous multi-accelerator architecture and a scheduler for multi-DNN workloads that combine compute- and memory-intensive sub-tasks in order to avoid the load imbalance between computation and memory-access tasks from different neural networks that jeopardize hardware utilization.

Chapter 3

Memory-Aware DNN Algorithm-Hardware Mapping via Integer Linear Programming

As discussed in Section 2.2.4, the task of efficiently mapping DNN layers onto DSAs is complex due to the enormous number of possible mapping configurations. The combinatorial explosion of choices makes exhaustive exploration infeasible, even with the aid of fast analytical simulation tools. This challenge is further intensified by the distinct constraints and requirements imposed by various DNN workloads and hardware architectures. To address these difficulties, a variety of mapping strategies have been proposed in the literature, leveraging heuristics, black-box optimization techniques, or mathematical models to efficiently explore the so-called mapspace, that is, the space of all possible mappings [111, 145].

This chapter introduces LEMON, a novel mapper based on integer linear programming (ILP) designed to map DNN layers onto DSAs. LEMON systematically incorporates the constraints and requirements of both the DNN models and the underlying accelerator architectures, with the primary goal of optimizing mapping decisions for performance and energy efficiency. The ILP formulation is memory-aware, explicitly modeling memory accesses, access energy costs, and bandwidth constraints. We evaluate the proposed mapper across a diverse set of DNNs and memory hierarchies, demonstrating its effectiveness and efficiency compared to state-of-the-art mapping methods. The reference implementation of LEMON is publicly available at `https://github.com/haimrich/lemon`.

This work was presented at the *20th ACM International Conference on Computing Frontiers*, held in Bologna in 2023, and published in the conference proceedings [188].

3.1 Methodology

In this work we formulate the mapping search as an ILP problem. The goal is to achieve a mathematical formulation that is flexible enough to represent different memory hierarchies, efficient enough to be solved rapidly and thorough enough to achieve near-optimal or optimal mapping solutions. We name the proposed model LEMON. Its main decision variables are binary and are divided in two categories: the first category encodes for loop boundaries, both temporal and spatial, and the latter encodes for loop permutations. In previous work, employing Mixed-Integer Programming (MIP) optimization, each binary variable encoded for a prime factor of a workload dimension [99]. In this work, binary variables encoding for loop bounds represent a divisor of a workload dimension. As explained in the following, this allows to accurately calculate the non-linear buffer utilization of the inputs feature maps without transforming the optimization problem in a non-linear one.

The objective of the optimization problem is explainable and consists of a linear combination of inference energy and latency. Hence, the designer can easily explore the trade-off and prioritize energy or latency by changing the multiplicative coefficient of the two terms, without dealing with more obscure metrics, such as buffer and MAC units utilization [99]. In order to calculate energy and latency for objective evaluation, one needs to calculate various expression starting from the binary decision variables.

Firstly, buffer utilization is calculated for each buffer in the memory hierarchy starting from decision variables encoding loop bounds. The buffer utilization is then taken into account together with decision variables encoding for loop permutations in order to calculate memory writes at each level of the hierarchy. Then, buffer reads at each level are derived considering buffer writes in the lower level and loop spatial unrollings. At this point, the total count of memory accesses for each buffer in the architecture can be calculated. The energy term is then calculated multiplying the memory access counts by the energy access costs derived querying Accelergy [229] estimation tool. The latency term is calculated considering the bandwidths of each memory in the hierarchy (including DRAM). For latency calculation, we adopt the same strategy of other analytical models [164] that, assuming double buffering, calculate not only the total inference latency as the maximum of the number of cycles needed by each buffer to access all the data at the given bandwidth rate, but also the compute cycles, obtained dividing the number of MAC operations of the workload by the number of utilized arithmetic units. This allows the optimizer to avoid requiring too many accesses to low-bandwidth buffers and incurring in memory-bound mappings. In the following, we describe the model formulation with constraints, intermediate expressions, and objectives.

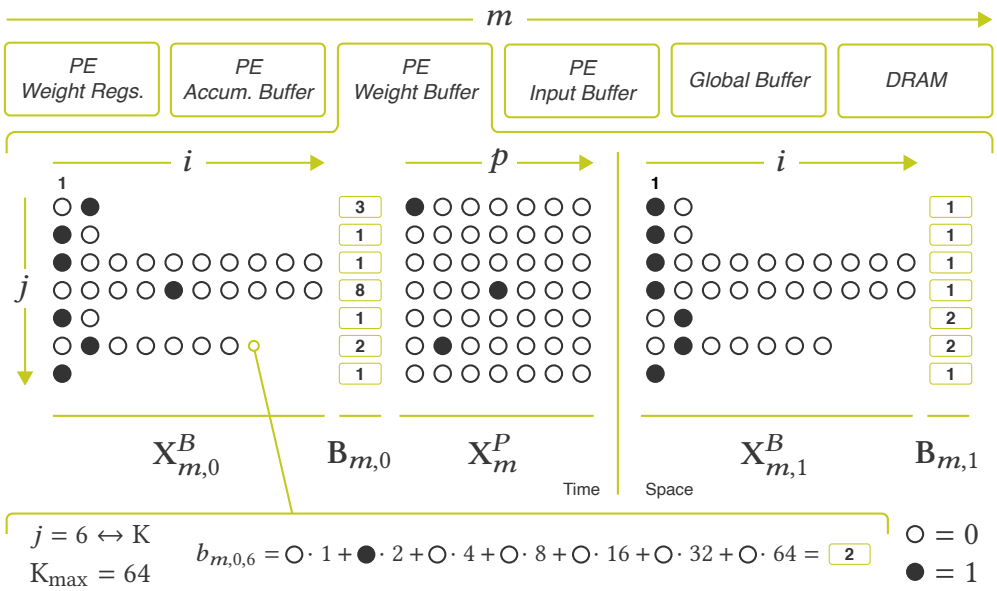


FIGURE 3.1: Main binary decision variables of the proposed ILP model, encoding for loop tiling and permutation.

3.1.1 Model Formulation

The binary decision variables involved in the proposed ILP model are summarized in Fig. 3.1, where each circle represents a decision variable. As anticipated, loop factorizations and permutations are encoded independently by two different groups of variables. For easier understanding Table 3.1 summarizes dimensions and indexes involved in following equations for the model formulation.

Symbol	Start	End	Description
m	1	M	Levels in memory hierarchy.
s	0	1	Temporal (0) and spatial (1) loops.
j	1	J	Dimensions of the workload.
i	1	σ_j	Divisors of workload dimension j.
p	1	J	Order rank of a temporal loop.
h	1	μ_t	Possible values for buffer utilization.
t	1	3	Workload datatypes.

TABLE 3.1: Indexes in model formulation

3.1.1.1 Loop Factorization and Spatial Unrolling

These first two mapping choices consist in choosing temporal and spatial loop bounds for each workload dimension and for each memory level. Thus, they are encoded in the model employing an array $X^B_{m,s,j}$ for each memory level m, for both temporal and spatial loops $s \in \{0, 1\}$ for each workload dimension j of binary variables $x^B_{m,s,j,i}$ for each possible value i that the corresponding loop bound can assume. Note that the number of possible loop bound values is equal to the number σ_j of integer divisors of the j-th workload dimension.

$$X^B_{m,s,j} = \begin{bmatrix} x^B_{m,s,j,1} & \cdots & x^B_{m,s,j,i} & \cdots & x^B_{m,s,j,\sigma_j} \end{bmatrix} \tag{3.1}$$

As a single bound value has to be assigned to each loop, $X^B_{m,s,j}$ has to be a standard unit vector that one-hot encodes the bound value. In the ILP model, this results in an equality constraint:

$$\sum_{i=1}^{\sigma_j} x^B_{m,s,j,i} = 1 \qquad \forall m, s, j \tag{3.2}$$

Thus, given the vector D_j of the divisors $d_{j,i}$ of the j-th workload dimension in ascending order, the value assumed by the corresponding loop bound $b_{m,s,j}$ can be written as:

$$b_{m,s,j} = X^B_{m,s,j} \cdot D_j^\intercal \tag{3.3}$$

As an example, considering a workload dimension of 4 and its divisors, a loop having a bound of 2, will be encoded as follows:

$$2 = \begin{bmatrix} 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} 1 & 2 & 4 \end{bmatrix}^\intercal$$

It can be noticed that $x^B_{m,s,j,1}$ always encodes for a loop bound of 1, which means that no factor is allocated in buffer m for the spacetime direction s and for the dimension j, while x^B_{m,s,j,σ_j} always encodes for the whole workload dimension.For these variables to represent a valid mapping, it is essential that for every workload dimension, the product of the corresponding loop bounds matches the dimension itself:

$$\prod_{m=1}^{M} \prod_{s=0}^{1} b_{m,s,j} = d_{j,\sigma_j}$$

This can be achieved with another linear constraint, leveraging the product rule of logarithms as follows:

$$\sum_{m=1}^{M} \sum_{s=0}^{1} X^B_{m,s,j} \cdot \log D_j^\intercal = \log d_{j,\sigma_j} \qquad \forall j \tag{3.4}$$

Similarly, the spatial unrolling at each memory level should not exceed the fan-out Fanout_m of each level m in the hierarchy, which can be calculated dividing the number of spatial instances at level m and the one at level $m-1$ (lower level). This constraint is again expressed by mean of a linear expression, considering only spatial loops:

$$\sum_{j=1}^{J} X^B_{m,1,j} \cdot \log D_j^{\intercal} \leq \text{Fanout}_m \qquad \forall m \tag{3.5}$$

In the end, the irregular matrix $\mathbf{X}^B$ of binary variables obtained considering the vectors $X^B_{m,s,j}$ as its rows and the linear constraints in Eqs. (3.2, 3.4, 3.5), encode for loop factorization and spatial unrolling.

3.1.1.2 Loop Permutations

The third mapping choice concerns the temporal loops. In fact, their ordering affects the number of accesses to each buffer. For each memory level m, loop permutations are expressed using a square matrix $\mathbf{X}^P_m$ of binary variables $x^P_{m,j,p}$ at first constrained to be a permutation matrix, using trivial linear constraints:

$$\sum_{j=1}^{J} x^P_{m,j,p} = 1 \quad \forall m, p \tag{3.6}$$

$$\sum_{p=1}^{J} x^P_{m,j,p} = 1 \quad \forall m, j \tag{3.7}$$

Therefore, if $x^P_{m,j,p} = 1$, then the temporal loop relative to the j-th workload dimension will be mapped at rank p. Please note that $\mathbf{X}^P_m$ is a square matrix because the number of possible ranks for each loop is equal to the number of workload dimensions J. The constraint expressed in Eq. (3.7) imposes the condition that the loop for each dimension j must be assigned to a rank p. For correct modeling of memory accesses, loops having unitary loop bounds should not be considered in subsequent calculations. For this reason the constraint in Eq. (3.7) has to be tweaked as follows:

$$\sum_{p=1}^{J} x^P_{m,j,p} = 1 - x^B_{m,0,j,1} \qquad \forall m, j \tag{3.8}$$

where $s = 0$, due to temporal loops and $i = 1$ because we check for unitary loop bounds. This new constraint avoids assigning a rank to loops having unitary bound. Finally, the matrix X^P_m together with constraints in Eqs. (3.6) and (3.8) encode for loop permutations at each memory level m.

3.1.2 Expressions

In order to calculate the energy and latency objectives starting from the binary decision variables just described, it is necessary to formulate some intermediate expressions.

		j						
		1: R	2: S	3: P	4: Q	5: C	6: K	7: N
t	1: Inputs	▲	▼	▲	▼	•		•
	2: Weights	•	•			•	•	
	3: Outputs			•	•		•	•

TABLE 3.2: Workload dimension relevancy for each datatype in a CONV/FC layer

3.1.2.1 Buffer Utilization

The calculation of data tile sizes that originate from loop factorization choices is needed to verify whether data can be fit in the available buffer capacities and avoid the generation of invalid mappings. Nevertheless, it is also necessary for the calculation of memory accesses. The computation of tile sizes has to be performed for each datatype involved in the workload, i.e., input features, weights and output features in DNN layers, considering relevant workload dimensions shown in Table 3.2. For weights and output features, data tiles allocated at level m can be easily calculated multiplying all the relevant loops (both temporal and spatial) allocated in all buffers at level $m' \leq m$. Thus, the buffer utilization for weights (2) and outputs (3) datatypes should be calculated as follows:

$$u_{m,t} = \prod_{m'=1}^{m} \prod_{s=0}^{1} \prod_{j=1}^{J} B_{m',s,j}^{o_{t,j}} \qquad \forall m, t \in \{2,3\} \tag{3.9}$$

Where $o_{t,j}$ is a constant equal to 1 when the j-th dimension is relevant for datatype t, as per Table 3.2, and 0 otherwise. We can linearize this expression, considering Eq. (3.3) and logarithm properties, obtaining:

$$\log u_{m,t} = \sum_{m'=1}^{m} \sum_{s=0}^{1} \sum_{j=1}^{J} o_{t,j} (X_{m',s,j}^{B} \cdot \log D_{j}^{\intercal}) \quad \forall m, t \in \{2,3\} \tag{3.10}$$

With this formulation, for buffers that contain exclusively weights or outputs, the capacity constraint can be easily formulated as follows.

$$\log u_{m,t} \leq \log \text{Capacity}_m \tag{3.11}$$

where Capacity_m is the number of elements that can be stored in the m-th buffer. However, there are two issues with this formulation: the first concerns the calculation of total buffer utilization when more than one datatype is stored in the same buffer; the second concerns the calculation of buffer utilization by the input feature maps. The former issue has been addressed in previous work assuming that shared buffers

are statically and heuristically partitioned [99]. This strategy can lead to sub-optimal mappings. Thus, it is required the calculation of either $\sum_{t=1}^{3} u_{m,t}$ or its logarithm in order to formulate the capacity constraint. As calculating the logarithm of a sum starting from the logarithm of the addends is not feasible without non-linear functions or costly approximations, we opt for the first option which involves calculating $u_{m,t}$ instead of its logarithm. This can be achieved introducing an array $X^U_{m,t}$ of binary variables $x^U_{m,t,h}$ one-hot encoding for each possible value that $u_{m,t}$ can take. Note that the number μ_t of possible values that $u_{m,t}$ can take is equal to the product of the numbers of divisors σ_j of each relevant workload dimension:

$$\mu_t = \prod_{j=1}^{J} \sigma_j^{o_{t,j}} \tag{3.12}$$

A set of linear constraints is needed to ensure that only one binary variable in the array $X^U_{m,t}$ is one and, most importantly, that it represents the correct value. We start by considering the Cartesian product of the divisors sets D_j of relevant dimensions:

$$\begin{aligned} H_t &= D_1^{o_{t,1}} \times \cdots \times D_J^{o_{t,J}} \\ &= \{(g_{h,1}, \ldots, g_{h,J}) \mid g_{h,j} \in D_j\}_{h=1}^{\mu_t} \end{aligned} \tag{3.13}$$

Then we add a set of linear constraints as follows:

$$\sum_{h=1}^{\mu_t} x^U_{m,t,h} \log g_{h,j} = \sum_{m'=1}^{m} \sum_{s=0}^{1} X^B_{m',s,j} \cdot \log D_j^{\intercal} \quad \forall m, t, j \tag{3.14}$$

for each memory level m, datatype t, and relevant workload dimension j. This will constraint some of the $x^U_{m,t,h}$ to be non-zero depending on the value taken by the product of lower level loop bounds for each relevant dimension. From the intersection of all these constraints, only one variable will end up being non-zero and will represent the actual buffer utilization for datatype t. To this end, we consider the precalculated array U_t of all possible values of $u_{m,t}$, keeping the elements of the Cartesian product H_t in the same order as considered for the constraints in Eq. (3.14). For weights and outputs U_t is defined as:

$$U_t = \left(\textstyle\prod_{j=1}^{J} g_{h,j} \mid g_h \in H_t\right) \quad \forall t \in \{2, 3\} \tag{3.15}$$

Thus, the buffer utilization can be calculated as follows:

$$u_{m,t} = X^U_{m,t} \cdot U_t \quad \forall m, t \tag{3.16}$$

The second issue related to the calculation of the occupation of the buffers by the inputs still remains to be addressed, but with this new formulation it is easy to model it without approximations or non-linear functions. It is simply required to calculate

appropriately the constant array U_1 for inputs. For convolutional layers, input tile size at level m is calculated as follows:

$$u_{m,1} = \check{W}_m \check{H}_m \check{C}_m \check{N}_m$$

where $\check{C}_m, \check{N}_m$ are the product of loop bounds for the respective workload dimensions in buffer levels $m' \leq m$ while $\check{W}_m$ and $\check{H}_m$ represent the tiled width and height respectively of the input feature map tile size and are calculated follows:

$$\check{W}_m = \text{Stride}_w(\check{P}_m - 1) + \text{Dilation}_w(\check{R}_m - 1) + 1$$

$$\check{H}_m = \text{Stride}_h(\check{Q}_m - 1) + \text{Dilation}_h(\check{S}_m - 1) + 1$$

Thus, U_1 can be defined considering the above expressions (instead of the product) and the possible values of the involved workload divisors from the cartesian product H_1. In this way, $u_{m,1}$ can be easily calculated. For inputs, it is also useful to calculate the logarithm of utilization as follows.

$$\log u_{m,t} = X^U_{m,t} \cdot \log U_t \quad \forall m, t \tag{3.17}$$

At this point, we have successfully obtained an exact estimation of all the buffer utilizations and their logarithms. We can proceed defining the general capacity constraint:

$$\sum_{t=1}^{3} u_{m,t} \leq \text{Capacity}_m \quad \forall m \tag{3.18}$$

Furthermore, buffer utilization estimations can now be considered for buffer memory access modeling.

3.1.2.2 Buffer Writes

For each datatype, buffer writes can be calculated considering the number of elements stored in the buffer (buffer utilization) and multiplying it by the number of times this stored data will change during the layer execution. The first term is already known as $u_{m,t}$, while the second which we refer to as $c_{m,t}$ is yet to be defined. At a higher level, this second term is calculated by multiplying all the temporal loop bounds allocated in buffers at levels $m' > m$ that are either relevant for datatype t or are outer than a relevant loop. In other words, the innermost relevant loop determines which loop bounds will have to be multiplied because it represents the relevant index that changes more frequently. Thus, $c_{m,t}$ has to be calculated considering loop permutations. We start by introducing for each datatype t, for each buffer level m and for each buffer $m' > m$ a

vector $X^R_{t,m,m'}$ of binary variables $x^R_{t,m,m',p}$ that we constrain to be 1 if the permutation level p hosts a relevant loop or any of the levels $p' < p$ and $m'' < m'$ does, and 0 otherwise.

$$\begin{aligned} x^R_{t,m,m',p} &\geq \sum_{j=1}^{J} x^P_{m',j,p} \cdot o_{t,j} && \forall t, m, m' > m, p \\ x^R_{t,m,m',p} &\geq x^R_{t,m,m',p-1} && \forall t, m, m' > m, p > 1 \\ x^R_{t,m,m',1} &\geq x^R_{t,m,m'-1,J} && \forall t, m, m' > m + 1 \end{aligned} \tag{3.19}$$

We now have a binary variable that indicates whether a loop at a certain permutation level should be multiplied. As variables encoding bounds are organized per workload dimensions, we define arrays $X^J_{t,m,m'}$ of binary variables $x^J_{t,m,m',j}$ that will be 1 when the loop relative to the j-th should be multiplied and 0 otherwise:

$$x^J_{t,m,m',j} = \sum_{p=1}^{J} x^P_{m',j,p} \cdot x^R_{t,m,m',p} \quad \forall t, m, m' > m, j \tag{3.20}$$

The above equation appears to be a quadratic constraint, but since the product involves binary variables it can be linearized introducing auxiliary variables and linear constraints [75]. At this point, we can easily calculate $c_{m,t}$ as:

$$\log c_{m,t} = \sum_{m'=m+1}^{M} \sum_{j=1}^{J} (X^B_{m,0,j} \cdot \log D^{\intercal}_j) \cdot x^J_{t,m,m',j} \tag{3.21}$$

Thus, the logarithm of $\text{writes}_{m,t}$ for buffer level m and the datatype t is defined as:

$$\log \text{writes}_{m,t} = \log c_{m,t} + \log u_{m,t} \tag{3.22}$$

Notice that this represent the number of buffer writes per spatial instance. The total number of $\text{writes}^{\text{tot}}_{m,t}$ can be calculated adding the logarithm of all the spatial loop bounds at level $m' > m$.

3.1.2.3 Buffer Reads

All elements that are written at buffer level m are supposed to be read from the first buffer containing the same datatype which is higher in the memory hierarchy. In absence of spatial fanouts the number of reads is equal to the number of writes. However, if the number of spatial instances is different between the communicating levels, the multicast (spatial reuse) opportunity arises and the number of reads is calculated considering the spatial unrollings. Thus, the number of writes is multiplied by the number of unicast

communications that are determined by the relevant spatial loops in the buffer level above (containing the same datatype).

$$\log \text{reads}^{\text{tot}}_{m,t} = \log \text{writes}_{m-1,t} + \sum_{j=1}^{J} (X^{B}_{m,1,j} \cdot \log D^{\intercal}_{j}) \cdot o_{t,j} \tag{3.23}$$

Buffer reads per instance can be calculated dividing by all the spatial loop bounds at levels $m' > m$. When the buffer is the closest to the MAC units the writes in the expression are replaced with number of MAC operations to be executed, because for each operation an operand will be read from this buffer.

3.1.3 Objective

The objective of the proposed MILP model consists in a linear combination of the energy and latency terms, calculated considering memory accesses.

$$\text{objective} = \alpha \cdot \text{energy} + \beta \cdot \text{cycles} \tag{3.24}$$

The two hyperparameters α and β can be adjusted according to the designer's need to successfully trade off the two metrics.

3.1.3.1 Energy

The energy term is calculated multiplying memory accesses for the average energy access costs that can be obtained using estimation tools such as Accelergy [229]. We define e_m as the energy access cost to the m-th buffer. In order to calculate the total energy term, calculating the total number of accesses is needed, i.e., summing writes and reads, but we only know their logarithms. Thus, we need to use the exponentiation function, which is non-linear. As this expression will not be subject to constraints it can be safely approximated. A frequent approach is to use piecewise-linear approximations of the non-linear function [116]. The higher the accuracy of the approximated function, the greater the complexity of the model. In our case, a piecewise-linear approximation $\tilde{\exp}(\cdot)$ of the exponentiation function is employed to calculate the number of total accesses to each buffer:

$$\text{accesses}_m \approx \sum_{t=1}^{3} \left(\tilde{\exp}\left(\log \text{reads}^{\text{tot}}_{m,t}\right) + \tilde{\exp}\left(\log \text{writes}^{\text{tot}}_{m,t}\right) \right) \tag{3.25}$$

Hence, the total energy expression is calculated as follows:

$$\text{energy} \approx \sum_{m=1}^{M} e_m \cdot \text{accesses}_m \tag{3.26}$$

3.1.3.2 Latency

The latency term is calculated as the maximum of the number of cycles required by each memory level to access all the elements required by the mapping and the compute cycles. The compute cycles are calculated by dividing the total number of MAC operations by the number of utilized MAC hardware unit instances (product of all the spatial loop bounds).

$$\text{cycles}_{\text{MAC}} \approx \tilde{\exp}\left(\log(\text{Num. MACs}) - \sum_{m=1}^{M}\sum_{j=1}^{J} X^{B}_{m,1,j} \cdot \log D^{\intercal}_{j}\right) \tag{3.27}$$

The cycles required at each buffer are calculated by dividing the total number of accesses by the bandwidth measured in elements per cycle, which is provided as input.

$$\text{cycles}_{m} \approx \sum_{t=1}^{3} \frac{\tilde{\exp}\left(\log \text{reads}_{m,t}\right) + \tilde{\exp}\left(\log \text{writes}_{m,t}\right)}{\text{Bandwidth}_{m}} \tag{3.28}$$

Finally, the latency term is calculated as follows:

$$\text{cycles} \approx \max\left\{\text{cycles}_{\text{MAC}}, \text{cycles}_{1}, \ldots, \text{cycles}_{M}\right\} \tag{3.29}$$

Note that the maximum operator can be linearized without approximations with the introduction of auxiliary continuous and binary variables and linear constraints [10].

3.2 Evaluation

The described ILP model is implemented and solved using Gurobi optimizer [84], which is capable of efficiently solving mixed-integer linear and quadratic optimization problems. We use Timeloop/Accelergy [164, 229] as a validated evaluation platform to estimate the energy and latency of optimized mappings on the chosen architecture. We first evaluate the proposed model, comparing the resultant optimized mappings against some state-of-the-art techniques in terms of Energy-Delay Product (EDP). In particular, we consider publicly available mappers based on different optimization techniques, i.e., genetic algorithms, gradient-based search and mixed-integer programming. All the compared methods are run on a machine featuring a AMD Ryzen 5900X 12-core CPU, 32GB of RAM and an Nvidia RTX 3060 GPU. Selected benchmark workloads consist in entire popular convolutional networks or some of their layers. For each comparison, we try to replicate the hardware architecture considered in the related paper, demonstrating the flexibility of the proposed method. Then, we test the degree of optimality of a solution found by LEMON by exhaustively testing all the possible mappings for a small layer. Finally, we explore some of the additional features of the proposed model concerning fixed-dataflow mapping, memory-bypass optimization, and static buffer partitioning. Table 3.3 shows the specifications of all target architectures considered in the following experiments.

TABLE 3.3: Memory hierarchies of target architectures considered in evaluations

Component	Num. Instances	Stored Datatypes	Size	Bandwidth
GAMMA				
MACs	1024×		16 bits	
Register File	256×	WIO	0.5 KB	∞
Global Buffer	1×	WIO	128 KB	∞
DRAM	1×	WIO	∞	∞
MindMappings				
MACs	256×		16 bits	
Input Reg.	256×	I	16 KB	2 GB/s
Weight Reg.	256×	W	28 KB	2 GB/s
P.Sum. Reg.	256×	O	20 KB	2 GB/s
P.Sum. Buffer	1×	O	16 KB	16 GB/s
Weight Buffer	1×	W	48 KB	16 GB/s
Input Buffer	1×	I	192 KB	16 GB/s
DRAM	1×	WIO	∞	∞
CoSA				
MACs	4096×		8 bits	
Weight Reg.	4096×	W	1 B	∞
P.Sum. Buffer	1024×	O	384 B	∞
Weight Buffer	1024×	W	4 KB	∞
Input Buffer	64×	I	8 KB	∞
Global Buffer	1×	IO	64 KB	16 GB/s
DRAM	1×	WIO	∞	17.9 GB/s
Other				
MACs	1024×		8 bits	
Local Buffer	1024×	WIO	8 KB	∞
Global Buffer	1×	WIO	64 KB	16 GB/s
DRAM	1×	WIO	∞	17.9 GB/s

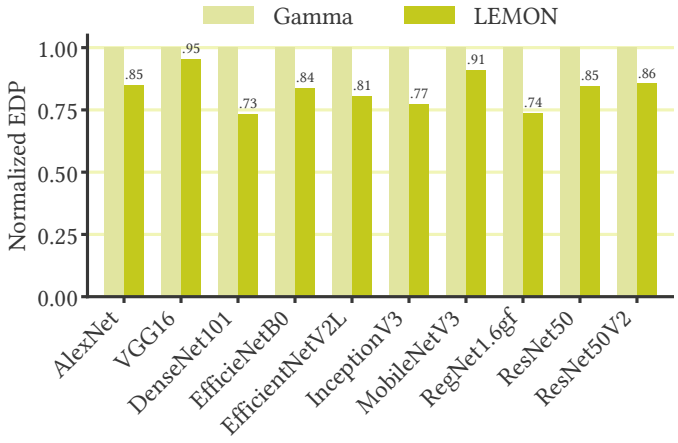


FIGURE 3.2: Comparison of inference EDP considering mappings produced by LEMON and solutions obtained using GAMMA.

3.2.1 Comparisons

Genetic Algorithm

We compare LEMON with a state-of-the-art mapper based on a genetic algorithm called GAMMA [109]. We consider entire convolutional networks, mapping each layer individually using the two approaches and summing energy and latency metrics. The target hardware architecture is the one presented in the GAMMA paper [109] and consists of a three-level memory hierarchy (DRAM, Global Buffer, PE Local Register) of buffers storing all datatypes. The number of generations and the population size configured for the genetic algorithm were 50 and 100, respectively, for each layer. Our experiments indicate that LEMON outperforms GAMMA in terms of both solution quality and computational efficiency. Specifically, we observed a reduction of the network inference EDP between 5% and 27%, as shown in Fig. 3.2, while simultaneously achieving between 1.1× and 19.1× search time speedup compared to GAMMA. These results demonstrate the potential of LEMON as a viable alternative to a genetic algorithm approach for solving the mapping problem.

Gradient-based Search

We also evaluate the proposed model against a gradient-based search approach called MindMappings [89]. This method requires the training of a surrogate neural network model considering a reference architecture and workload type (such as convolutional layer). Training is carried out by building a dataset from sampled random mappings evaluated using an analytical tool, such as Timeloop, or on the target hardware. Thus,

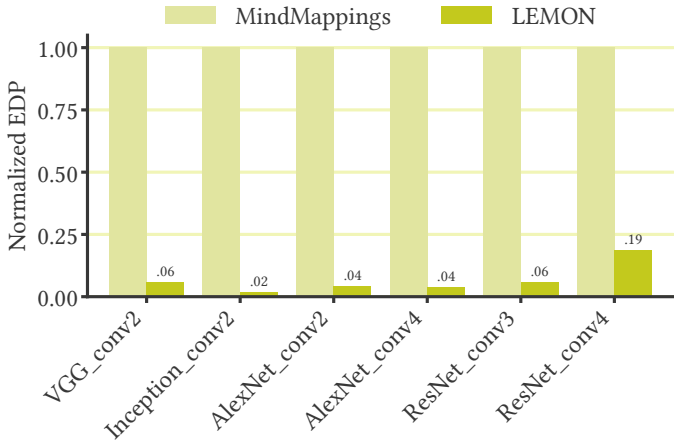


FIGURE 3.3: EDP comparison of mappings produced by LEMON and MindMappings.

changing target architecture requires time-consuming dataset building and model retraining. For our evaluations, we considered the publicly available pre-trained model and the corresponding architecture described in the paper and ran MindMappings gradient-based search for 1000 iteration steps on each layer. Furthermore, the available model was trained for layers without stride and dilation. Thus, we considered only the convolutional layers mentioned in the original paper as benchmark. The results show that LEMON outperforms MindMappings in terms of energy and latency savings. Specifically, we observed a reduction in EDP ranging from 81% to 98% when using LEMON, compared to MindMappings, as shown in Fig. 3.3.

3.2.1.1 Other MIP Approach

As LEMON is not the first mapper that uses an MIP approach, we compare it against another state-of-the-art MIP-based mapper, namely CoSA [99]. We consider the Simba-like [200] chiplet hardware architecture described in the paper, additionally we consider DRAM and Global Buffer bandwidths. The results show that LEMON tends to achieve better mappings in terms of EDP across different neural network workloads. In particular, Fig. 3.4 shows EDP savings ranging from 33% to 83%. These results should be mainly attributed to LEMON's greater cognition of memory bandwidths, which allows it to avoid bottlenecked memory-bound mappings, and of access energy at each level of the memory hierarchy. Furthermore, these results highlights the potential of LEMON as a powerful tool for solving the mapping problem in a fabricated hardware accelerator such as Simba.

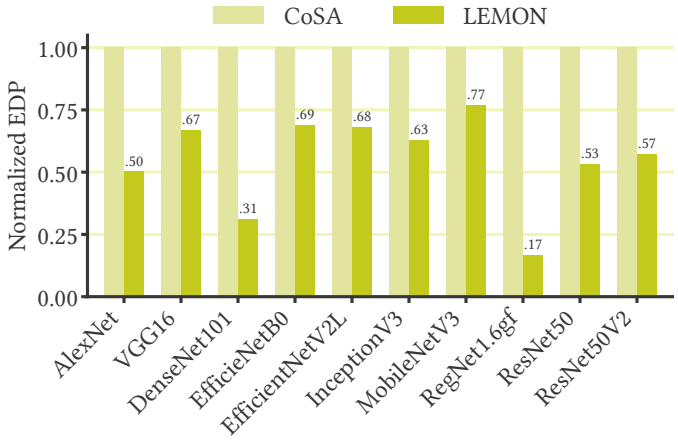


FIGURE 3.4: Inference EDP comparison between mappings produced by LEMON and solutions obtained using CoSA.

3.2.2 Optimality

Assessing the solutions attainable through the proposed method in comparison to the optimal mapping for a specific hardware architecture and network layer is highly challenging, as it requires an exhaustive exploration of the mapping space. This exploration is often unfeasible due to the vast number of possible combinations, particularly for complex levels and architectures. Therefore, we can only evaluate optimality considering a very simple level, i.e., the first layer of AlexNet and a similarly simple architecture, namely the last one described in Table 3.3 but featuring only 64 MAC units (one per PE). We evaluated all possible mappings using Timeloop and compared the best one found (optimal) with the mapping produced by LEMON for the same specifications. LEMON produced a mapping with an EDP 1.08% higher than the optimal solution, but required 17500× less search time compared to Timeloop exhaustive mapper (taking only a few seconds versus several hours).

3.2.3 Fixed Dataflow and Static Partitioning

So far we have assumed that the target architectures are capable of executing layers according to mappings that manifest different dataflows and that they can be reconfigured to do so at runtime. This hardware flexibility can come at the cost of increased area or increased power. For example, in [128] the authors highlight an energy consumption higher by about 11% in a reconfigurable dataflow accelerator (MAERI) compared to one with weight stationary dataflow (NVDLA) in particular scenarios. The dataflow is mainly determined by loop permutations. Depending on the use case, it may therefore be useful to have all layers of a network run according to mappings that expose the same

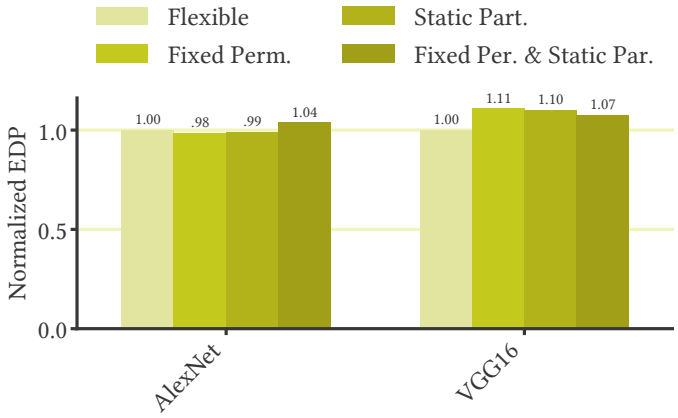


FIGURE 3.5: Normalized EDP of mappings found using LEMON in flexible, static partitioning and fixed permutation modes.

loop permutations. However, such permutations may not be known a priori. For this purpose, LEMON can be run in a mode that allows to optimize the mapping of all layers of a network at the same time, constraining the equality of some specifications across all layers. Similarly, the partitioning of buffers that can accommodate multiple datatypes can be constrained because the designer may want the fraction allocated to each datatype to remain constant during network execution for implementation reasons (e.g., address generation). Fig. 3.5 shows a comparison of the EDP metric for the mapping obtained through optimization with complete flexibility, optimization fixing permutations only, fixing partitioning only, and finally fixing both permutations and partitioning, for different networks. This comparison highlights the trade-offs between flexibility and energy efficiency in the mapping process, and allows the user to make informed decisions based on their specific application requirements. For example, for the networks considered, the EDP degradation from constraints is never higher than 11% and flexibility results in better EDP, but note that we have not considered the energy and latency overheads, e.g., 11% more energy in MAERI [128]. It is worth noting that while running optimization for flexible mappings is fast and can take only a few minutes, running optimization for a whole network and fixing mapping specifications can take several hours due to the complex interdependence of decision variables. However, this trade-off may be justified if building a more flexible architecture would result in higher costs.

3.2.4 Buffer Datatype Bypass

In previous experiments, the datatype storage capabilities of each buffer in the architecture were assumed to be known. However, LEMON also features an optimization mode that utilizes a more complex quadratic model to optimize bypass directives. This

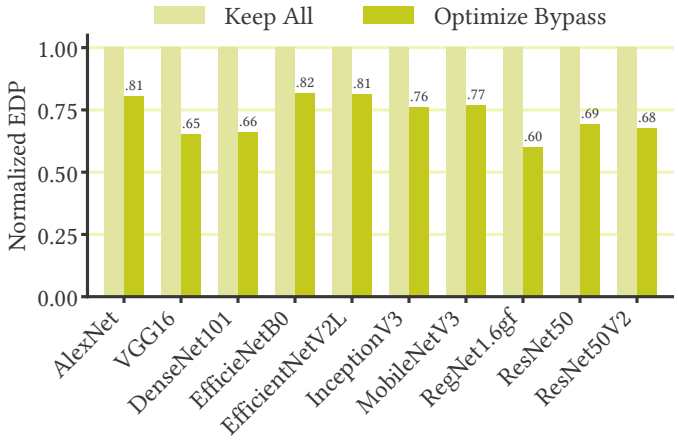


FIGURE 3.6: Effect on EDP reduction of LEMON bypass optimization in a 2-level architecture for different networks.

mode involves the introduction of binary decision variables that indicate the presence or absence of a datatype in each buffer. Fig. 3.6 shows the mapping metrics obtained when each buffer in the reference architecture can store all datatypes, and then the results from the optimized bypass. It can be seen that the reduction in EDP when optimizing bypass can reach up to 40%.

3.3 Conclusion

We introduced LEMON, an ILP-based DNN layer mapper that can generate near-optimal mappings quickly and efficiently by accurately modeling memory accesses and the corresponding energy demands and bandwidth requirements across a broad range of memory hierarchies. Further research should focus on overcoming the limitations of our approach, including absence of data sparsity awareness [230], lack of imperfect factorization capabilities [96], inability to generate uneven mappings [144] and absence of cross-layer data movement consideration [215].

Chapter 4

Multi-DNN Multi-Accelerator Multi-Objective Design Space Exploration via Genetic Algorithms

Deep Neural Network (DNN) accelerators have become foundational to modern computing systems, powering diverse applications from edge devices [78] to large-scale cloud data centers [77]. The unprecedented growth in demand for AI workloads has led to the deployment of increasingly larger and more complex DNN accelerators. To address scalability and adaptability requirements, recent research has shifted towards domain-specific architectures (DSAs) that combine multiple accelerators—often as heterogeneous chiplets—into expansive multi-accelerator systems.

A central enabler for scalability in these systems is support for *multi-DNN workloads*, where several DNN models are executed concurrently. This paradigm is especially critical in multi-tenant data center environments, enabling efficient resource sharing and service to diverse inference requests, as discussed in Section 2.3.4. At the edge, multi-DNN execution underpins complex, integrated tasks. However, the design and optimisation of such multi-accelerator systems introduce significant challenges, particularly as the design space for both hardware configurations and workload mappings grows combinatorially with system heterogeneity and workload diversity.

To better understand the current landscape, Table 4.1 summarises the state-of-the-art in multi-accelerator system design, comparing recent works along four key axes:

- **Hetero-Core:** Whether heterogeneous core or chiplet integration is supported.
- **Multi-DNN:** Support for concurrent multi-DNN workloads.
- **HW-MP:** Availability of co-optimisation between hardware configuration and mapping strategy.

- **Multi-Obj:** Capability for multi-objective design space exploration (DSE).

Work	Hetero-Core	Multi-DNN	HW-MP	Multi-Obj
Simba [200]	✗	✗	✗	✗
TPUv4 [77]	✗	–	✗	✗
CS-2 [30]	✗	✗	✗	✗
AI-MT [11]	✓	✓	✗	✗
PREMA [40]	✓	✓	✗	✗
Planaria [73]	✗	✓	✓	✗
Herald [128]	✓	✓	✓	✗
VELTAIR [140]	✗	✓	✗	✗
MAGMA [110]	✓	✓	✗	✗
H3M [237]	✗	✓	✓	✗
MOHaM [49]	✓	✓	✓	✓

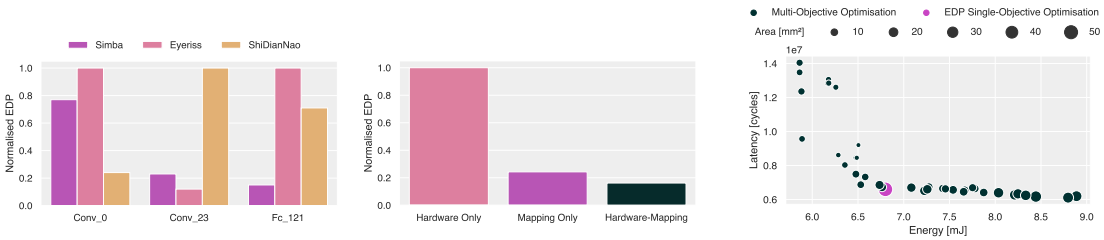
TABLE 4.1: Comparison of multi-accelerator systems based on the availability of heterogeneous cores (Hetero-Core), multi-DNN workloads (Multi-DNN), hardware-mapping co-optimisation (HW-MP), and multi-objective exploration (Multi-Obj).

As shown in Table 4.1, while recent advances have introduced either heterogeneous integration, multi-DNN support, or partial co-optimisation, no prior system comprehensively addresses all these requirements or provides a practical multi-objective DSE framework.

The necessity for holistic design is further illustrated in Figure 4.1, which highlights the need for (a) heterogeneity, (b) co-optimisation, and (c) multi-objective exploration in future multi-accelerator architectures.

Motivation and Challenges

The performance and efficiency of DNN accelerators are governed by two intimately connected factors: the hardware configuration and the mapping strategy for workloads. While the design spaces for these factors are each vast, their interdependence means that independent optimisation can lead to suboptimal system performance—especially as heterogeneity and multi-tenancy become the norm. Prior works, such as MAGMA [110], have demonstrated that the mapping space alone can be astronomically large (e.g., $O(1e81)$), rendering exhaustive search infeasible. Furthermore, the presence of diverse DNN workloads and heterogeneous accelerator templates introduces non-trivial trade-offs between key objectives such as latency, energy, area, and even higher-level concerns like total cost of ownership or carbon footprint.



(A) (B) (C)

FIGURE 4.1: Motivation: (a) Need for heterogeneity, (b) hardware-mapping co-optimisation, (c) multi-objective exploration in multi-accelerator systems.

Traditional DSE approaches often collapse multiple objectives into a single metric, potentially masking trade-offs and limiting the customisability of the resulting designs. This limitation becomes particularly acute in heterogeneous, multi-accelerator settings, where conflicting objectives are common and aggregation can obscure optimal solutions for specific deployment needs.

Contributions of This Chapter

To address these challenges, this chapter introduces **MOHaM** (Multi-Objective Hardware-Mapping co-optimisation), the first open-source framework for comprehensive DSE of multi-accelerator systems executing multi-DNN workloads. MOHaM enables:

- **Adaptability:** Supports heterogeneous accelerator templates for flexible system composition.
- **Scalability:** Efficiently handles large-scale, multi-DNN workloads with multi-tenancy.
- **Performance:** Jointly co-optimises hardware and workload mapping for superior system efficiency.
- **Solution Variety:** Delivers Pareto-optimal sets of designs, exposing trade-offs between latency, energy, area, and other objectives.

Specifically, MOHaM leverages the NSGA-II [51] multi-objective genetic algorithm, enhanced with custom operators geared towards this unique design problem, and integrates the Timeloop [164] and Accelergy [229] cost models for accurate evaluation. Through strategic sampling and efficient exploration, MOHaM delivers orders-of-magnitude speedup over exhaustive search techniques. Experimental results demonstrate that

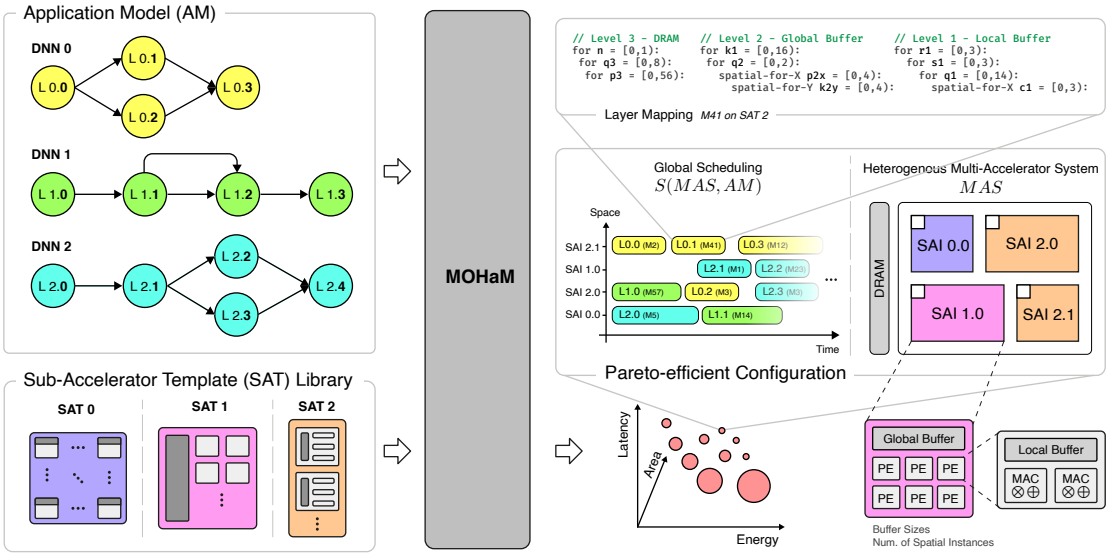


FIGURE 4.2: Overview of the MOHaM framework

MOHaM-designed systems achieve substantial improvements over state-of-the-art baselines, reducing latency and energy consumption significantly.

The research presented in this chapter is based on the work published in the *IEEE Transactions on Computers* in 2024 [49], the reference implementation is publicly available at `https://github.com/haimrich/moham`.

4.1 Overview

In this section, we describe the input and output to the MOHaM framework, as well as the general formulation of the problem.

4.1.1 Input

A DNN model is a directed graph where the nodes are layers and the edges are the dependencies between layers. A layer has a set of properties such as the type of layer (e.g., convolution, fully connected, transposed convolution, etc.) and the shape of input and weight tensors (e.g., width, height, channels).

An Application Model, AM, is a set of DNN models. The DNN models within an AM are assumed to be independent of each other and thus can be executed in parallel. For example, AM for an Augmented Reality (AR) application will have multiple DNN models, each with a specific task like object detection, gesture identification, depth estimation, etc. Figure 4.2 shows an AM formed by three DNN models.

A Sub-Accelerator Template (SAT) is a parameterised and reconfigurable DNN accelerator supporting different hardware configurations as well as mapping strategies. For example, a modifiable Simba chiplet [200] could be a SAT where the free parameters are buffer sizes, number of PEs and number of MAC units per PE and dataflow is weight-stationary.

As shown in Figure 4.2, the inputs to the MOHaM framework are the Application Model AM and a library of SATs.

4.1.2 Output

A Sub-Accelerator Instance (SAI) is an instance of a SAT configured with specific values for its parameters. For example, Simba SAT becomes a Simba SAI when its configurable parameters are given values, like, 128 PEs, 256 KiB GB, 32 KiB LB, weight-stationary dataflow, etc. SAI are interconnected through a Network-on-Package (NoP) and arranged in a 2D grid of tiles. A Multi-Accelerator System, MAS, consists of a $SSAI$, i.e., the set of heterogeneous SAI chiplets, and a SMI, i.e., the set of Memory Interfaces (MIs) for external HBMs/DRAMs. SAIs and MIs are all connected together by the NoP with a grid-like topology arranged in tiles. The MAS also defines the placement of each SAI in the NoP mesh tiles. Fig. 4.2 shows a MAS created by four SAIs and an external memory, interconnected by a NoP. The SAIs are instances of three parameterized and reconfigurable SATs, as shown in Fig. 4.2. For example, $SAI2.0$ and $SAI2.1$ are two different instances of $SAT2$.

The schedule S defines the space-time assignment of each layer in the AM on a SAI for execution. The MOHaM schedule also includes information about the mapping according to which each layer will be executed. The schedule should also respect the dependencies of each DNN model in the AM, thus each layer should always be scheduled after all its dependencies. For example, in Fig. 4.2, shows the Gantt chart of a possible scheduling. The layer $L0.0$ is assigned to the $SAI2.1$ and will be executed according to mapping $M2$ which is a valid mapping for the $SAT2$. Note that, for example, the layer $L0.3$ has to wait for the execution of layer $L0.2$ to finish.

The output of the MOHaM framework is a Pareto set of design points, each consisting of a pair $\langle MAS, S \rangle$. Each Pareto-efficient configuration defines the multi-accelerator architecture (with specific SAIs, their template, their hardware parameters and their placement in NoP tiles) and the layer global schedule on this architecture.

4.1.3 Problem Formulation

Given an application model, AM, and a library of heterogeneous, parameterised and reconfigurable sub-accelerator templates, SATs, find the Pareto-optimal set of multi-accelerator systems, MAS, and an optimal schedule, S, for each one of them to minimise the overall system latency, energy and area.

4.2 Inner Workings

To search the enormous design space, the proposed MOHaM framework adopts a two-step approach, *layer mapping* and *global scheduling*. In the first step, each layer of the AM is mapped onto each SAT available in the input library using MEDEA [190], a multi-objective single layer mapper. Then the Pareto-optimal set of mappings found for each layer is used in the second step to search the global scheduling. MOHaM is written in C++ using OpenMP [32] and uses the Timeloop/Accelergy [164, 229] framework as a simulation platform in both the steps. Algorithm 1 presents a high-level flow of MOHaM, and the following subsections describe its working in detail.

Algorithm 1 MOHaM High-Level Flow

```
Procedure LayerMapper(AM, SSAT)
    MG ← {}
    foreach layer in UniqueLayers(AM) do
        ML ← {}
        foreach arch in SSAT do
            MF ← RunMedea(layer, arch)
            ML.append(MF)
        MG.append(ML)
    return MG

Procedure GlobalScheduler(AM, SSAT, MG)
    PP ← InitialPopulation()
    for g ← 1 to G do
        OP ← ApplyCrossoverOperators(PP)
        OP ← ApplyMutationOperators(OP)
        OP ← Evaluate()
        MP ← FastNonDominatedSorting(PP, OP)
        OP ← Survival(MP)
    return PP.GetParetoEfficientIndividuals()

Procedure MOHaM(AM, SSAT)
    MG ← LayerMapper(AM, SSAT)
    ST ← GlobalScheduler(AM, SSAT, MG)
    return ST ;                          // Return Pareto Set of Schedulings
```

4.2.1 Layer Mapper

Let $SSAT$ be the set of SATs available in the input library. If each layer of the AM is mapped on each of the SATs, the search space can be as large as $|L| \times |SSAT|$. However, two layers, $l_i, l_j \in L$ can be instances of the same workload, i.e., have the

same problem dimensions. Hence, in an attempt to reduce the layer mapping search space, MOHaM only maps the unique layers to each of the SATs.

Let $M_{l,f,i}$ be a mapping for a layer $l \in L$ in an SA template $f \in SSAT$. $M_{l,f,i}$ is a mapping, i.e., a choice of a specific loop tiling, loop ordering, and paralellism for l on f. Now, let $MF_{l,f}$ be the Pareto-optimal set of mappings with respect to latency, energy and area for l in f. $MF_{l,f}$ can be represented by:

$$MF_{l,f} = \{ M_{l,f,i} \mid i = 0, \ldots, (m_{l,f} - 1) \} \tag{4.1}$$

where $m_{l,f}$ is the number of Pareto-optimal mappings. Now, let ML_l be the set of Pareto sets of mappings obtained for l for each of the input SA templates of $SSAT$. ML_l can be represented by:

$$ML_l = \{ MF_{l,f} \mid f = 0, \ldots, (F - 1) \} \tag{4.2}$$

where, F is the number of input SA templates, i.e., $|SSAT|$. Finally, we can define the set MG as:

$$MG = \{ ML_l \mid l = 0, \ldots, (L - 1) \} \tag{4.3}$$

where, L is the number of layers in the AM. To obtain the Pareto-optimal sets in Eq. (4.1) through Eq. (4.3), MOHaM leverages the MEDEA [190] infrastructure. With respect to the optimisation objectives, latency, energy and area, MEDEA searches for a Pareto-optimal set of mappings for a layer on a SAT employing a GA with custom operators.

4.2.2 Global Scheduler

The global scheduler has a similar role to the negotiator of EPOCA [194] (a multi-objective single model end-to-end mapper and architecture exploration tool) and is based on the NGSA-II [51] algorithm too. However, its purpose is different and more complex. The global scheduler of MOHaM returns a Pareto-optimal set of MAS and an optimal S for each of them with minimum latency, energy, and area. The scheduler uses the original selection and survival phases of the NGSA-II. However, multiple custom genetic operators are implemented for problem-specific crossover and mutation, and also for increasing the sampling efficiency to find better individuals in less time.

Chromosome Encoding

One of the most fundamental and important steps in employing a GA is to define an encoding for the individuals in the population. MOHaM requires this encoding to represent: mapping strategy of each layer, SAI of each layer, execution sequence of layers in the AM, SA template of each SAI, and NoP tile of each SAI. Therefore, as shown in Fig. 4.3, the global scheduler uses a chromosome that consists of two parts:

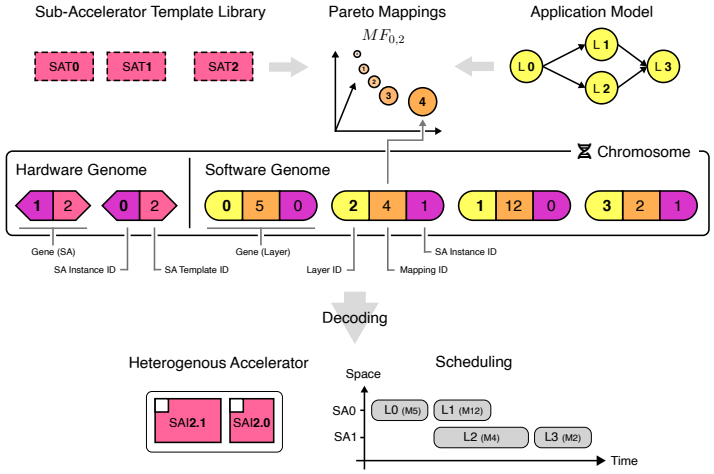


FIGURE 4.3: MOHaM global scheduler chromosome structure

- **Software Genome:** It encodes information about the layers of the AM. In general, we adopt a layer-centric approach, i.e., most of the information is associated with layers in the software genome. This part of the chromosome is an array of genes, where each gene denotes a layer. So, the number of software genes is equal to the number of layers in the AM. Each gene is a tuple $\langle LI, MI, SAI \rangle$ where LI is the layer identifier, MI is the mapping identifier and SAI is the SA instance the layer will be executed on. The order of the genes is a valid topological sorting[1] of the AM and represents the temporal sequence in which the layers will be executed.

- **Hardware Genome:** It encodes the instances SAI of the $SSAT$. It is also a list of genes, where each gene denotes a SA instance. Each gene is a tuple $\langle SAI, SAT \rangle$ where SAI 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 2D mesh. The number of genes is equal to the number of SA instances and varies between 1 and the maximum number of sub-accelerators that can be instantiated, specified in the search settings of MOHaM.

Custom Genetic Operators

Each chromosome must respect certain constraints for it to represent a valid individual. For example, the topological sorting of layers in the software genome must be valid. It has to be a traversal where a layer is placed only after all its dependencies. Similarly, the SAI

[1] A topological sorting of a directed graph is a linear ordering of its nodes where each node appears before all the nodes it points to.

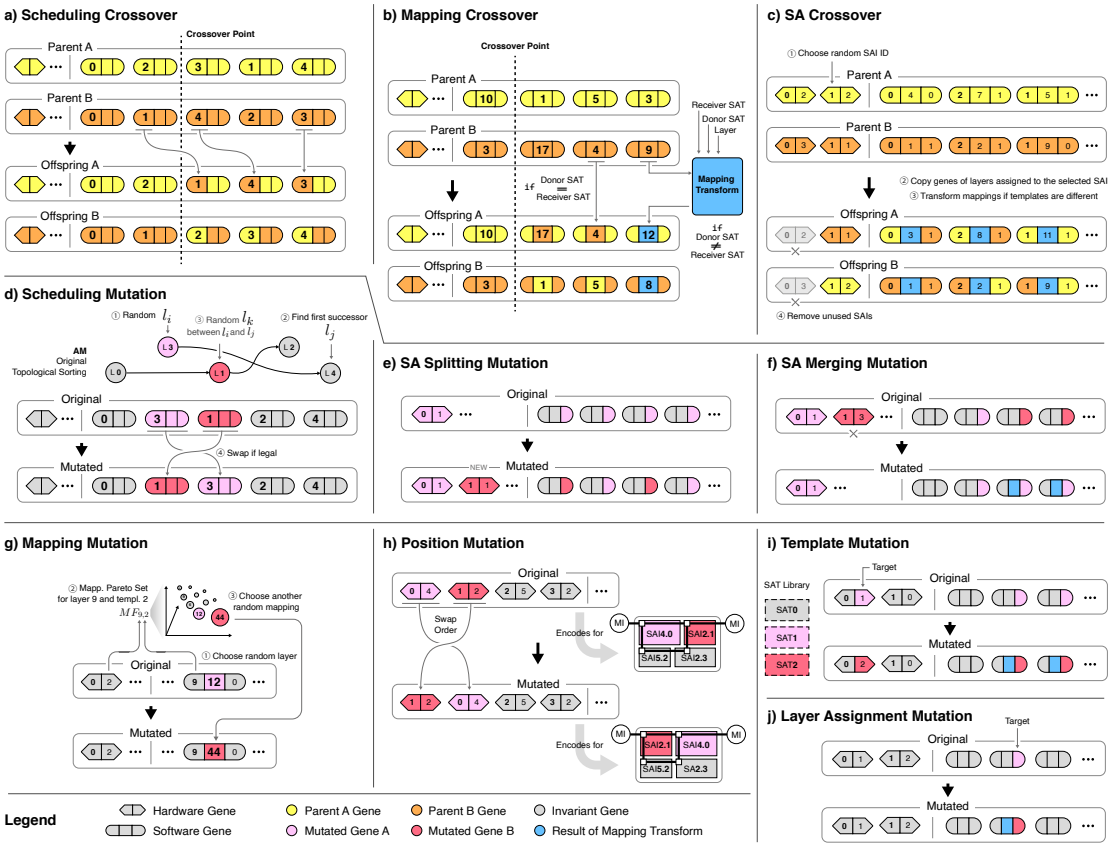


FIGURE 4.4: MOHaM-specific genetic operators

of a layer must refer to one of the encoded genes in the hardware genome. Furthermore, the MI of a layer must refer to one of the mappings in the $MF_{l,f}$ in Eq. (4.1). In order to improve the search efficiency, a set of custom genetic operators are implemented that considers these constraints. These operators either avoid generating invalid combinations or use compensation mechanisms. Operators targeting the topological sorting in the software genome are similar to [232]. MOHaM-specific genetic operators are shown in Figure 4.4 and are described here:

- **Scheduling Crossover:** It combines the topological sorting of the parent chromosomes, as shown in Figure 4.4a. It basically consists in generating an offspring by taking the first part of one of the parents, i.e., all the genes before the crossover point, and then appending all the genes from the other parent that are not present in the first part. It can be proved that by proceeding in this way, the two resulting offspring still represent valid topological sortings.

- **Scheduling Mutation:** It mutates the topological sorting of a chromosome, as shown in Figure 4.4b. Let L_i be a random gene (layer) in the software genome. Let L_j be the nearest layer in the topological sort that is dependent on L_i. Let L_k be a random gene between L_i and L_j. If all the layers on which L_k has a dependency lie before L_i in the topological sort, their position can be swapped. It can be proved that the resulting mutated software genome is still a valid topological sort.

- **Mapping Mutation:** It modifies the MI of a random layer l to mutate a chromosome, as shown in Figure 4.4c. Each gene in the software genome also encodes the mapping according to which the associated layer must be executed. The gene also establishes on which sub-accelerator the layer l will be executed. The SA is an instance of a particular architectural template f. Therefore, the possible mappings for the layer will be those in the set $ML_{l,f}$ obtained from the layer mapper step of the MOHaM algorithm. The Layer Mapping Mutation is achieved choosing a different possible mapping for a random layer l in the software genome.

- **Mapping Crossover:** It combines the mappings of the parent chromosomes, as shown in Figure 4.4d. It generates offspring by taking the mappings of layers from the first part of one of the parents, i.e., from all the genes before the crossover point, and the remaining ones from the other parent. However, the mapping for a layer (gene) in the offspring might not be valid if the SA instance is of a different template. In such a scenario, a compensation mechanism is required to generate valid offspring. As shown in Figure 4.4(d), a *Mapping Transform* mechanism is applied to find the most similar one among the possible mappings for the layer in that SA instance. This mechanism is also applied in other custom operators that require it. The idea behind this transformation is that a specific mapping can happen associated with a layer, for one or more of the energy, area and latency metrics that come with it. For this reason, the transformation aims to find the most "similar" mapping to the donor one in the space of the possible mappings in the receiver template. We normalise the metrics of all the mappings receiver set considering their respective variation ranges. We do the same for the donor mapping considering the variation range in the donor mapping set. Then we translate the donor mapping 3D point in the space of normalised mapping points of the receiver, and we pick the nearest neighbour as our result mapping.

- **SA Crossover:** It swaps a random SA instance s_i between the parent chromosomes, as shown in Figure 4.4e. If both parents A and B have an instance with the same identifier s_i, they are swapped and two offspring are generated. The layers assigned to the crossed accelerators in both offspring will be the union of the layers assigned to s_i in the parent A and s_i in parent B. This means that, in the offspring, some SAs will "loose" some layers, hence they could remain empty and unused and thus be removed. If s_i of A and s_i of B are of different SA templates, all layers

mappings that experience a template change undergo mapping transformation. If s_i is only in one of the parents, it is added to the other parent with all its assigned layers. In this case, only one offspring is generated.

- **SA Splitting Mutation:** It reduces the load in a random SA instance s_i to mutate a chromosome, as shown in Figure 4.4f. Another instance s_j of the same SA template is attached to the hardware genome. Subsequently, half of the layers currently assigned to s_i are randomly chosen and assigned to s_j. The goal is to increase parallelisation by increasing the number of sub-accelerators.
- **SA Merging Mutation:** It increases the load in a random SA instance s_i to mutate a chromosome, as shown in Figure 4.4g. Another instance s_j is chosen randomly, and all the layers currently assigned to it are assigned to s_i. If s_i and s_j are of different SA templates, all the mappings of imported layers undergo mapping transformation. The goal is to reduce chip area cost by reducing the number of SAs.
- **SA Position Mutation:** It swaps the position of two SAI hosting tiles in the NoP 2D Mesh, as shown in Figure 4.4h. The goal is to find a configuration where the system bandwidth is distributed among the NoP links and memory interfaces in a way to avoid bottlenecks and stalls during the schedule execution.
- **SA Template Mutation:** It modifies the SAT of a random SA instance s_i to mutate a chromosome, as shown in Figure 4.4i. All the mappings of the layers assigned to the chosen SA s_i undergo the mapping transformation function.
- **Layer Assignment Mutation:** It mutate the SAI assignment of a random layer, as shown in Figure 4.4j. If the destination SA has a different template from the original one, the mapping transformation is applied.

4.2.3 Objectives Evaluation

Three objective metrics are evaluated for each individual: makespan (latency), energy, and area. Both parts (i.e., software and hardware) of an individual's chromosome contribute to determining the value of these metrics. For example, by increasing the number of SA instances, the latency decreases and the occupied area increases. Similarly, the mapping strategy for each layer of the AM, and the SA templates affect all three metrics. Scheduling determines the latency. The position of SAI hosting tiles in the NoP 2D Mesh determines the presence of bandwidth bottlenecks, which in turn determines latency. The translation from chromosome encoding to target metrics does not take place by means of analytical formulas, but by processing and combining the results from the Timeloop/Accelergy [164, 229] framework, which allows the simulation of a single layer on a single accelerator instance.

The translation process begins from the hardware genome. Each hardware gene is converted into a SA instance of the appropriate template. At this point, the size of buffers and the number of spatial instances of each SA is still unknown. Then software genomes are examined to identify which mappings are used to map each gene (layer) to an SA instance. The size of each buffer of an SA instance is set to the maximum required by the mappings of all the layers assigned to it (similarly to the negotiation mechanism in EPOCA). The same is done for the number of PEs and all other free parameters. At this point, each SA instance can execute all the assigned layers, and the area metric can be calculated. The total area of the system will be the sum of all the SA instance areas.

As for the estimation of energy, since each mapping is associated with an energy metric, it is possible to calculate the total energy by adding the value for all the layers, but this is not enough. In fact, the energy of reading and writing from the buffers depends on their size and, therefore, must be calculated for each mapping on the SA instance with the updated parameters that gets assigned. At this point, the correct energy can be obtained by adding these assigned values.

The latency is calculated starting from the topological sorting encoded in the software genome. The genes of the software genome are read sequentially, the corresponding layer is mapped on the SAs to which it is assigned. By proceeding according to the topological sorting, it is guaranteed that by the time a layer is ready to be scheduled, its dependencies have already been scheduled. Its starting time will be the maximum between the end time of the last layer assigned to the same SA and the maximum end time of all its dependencies. The end time of the layer will be calculated summing the latency of the layer mapping to the start time. The latency will be the latest end time among all the layer. However, this is true only when there are no communication or memory bottlenecks.

Communication Modelling

Before concluding the latency evaluation, the memory and communication models are taken into account. In MOHaM, SAs and main memory are considered to be interconnected using a 2D mesh NoP. It is assumed that the entry points of the NoP are the so-called Memory Interfaces (MIs). A MI could be associated with a main memory bank. Each SA retrieves data from the nearest MI and writes the output feature map to the same MI. It is assumed that some CPU processing happens between accelerated layer execution (e.g., tensor reordering) and that the output of this processing is stored in the memory bank nearest to the SA that will execute the next layer. How this happens is outside the scope of this work. Multiple SAs, depending on their position in the NoP, can be assigned to the same MI competing for the link. The set of SAs is partitioned into clusters based on the MI to which it belongs. When, in a time segment where the execution of several layers is concurrent, the required bandwidth is less than the minimum between the bottleneck link bandwidth and MI, this segment undergoes

Table 4.2: Multi-DNN workload scenarios

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

a temporal dilation. However, this expansion only concerns the SAs belonging to the same MI cluster and requires compensation of the starting times of all subsequent layers to keep dependencies respected. After the evaluation of all the time segments in which the stalls occur, the maximum end time among all the levels corresponds to the actual latency.

Convergence Criterion

The MOHaM framework supports a GA stopping criterion based on the density of non-dominated solutions presented in [186]. Alternatively, simulation can also be configured to run for a fixed number of generations.

4.3 Experimental Results

This section presents multiple experiments conducted to evaluate the performance of the proposed MOHaM framework. Inspired by the popular MLPerf benchmark suite [180], multiple DNN models from major application domains like vision, language, and recommendation are considered. Although a multi-accelerator system is usually employed in the cloud, i.e., data centers, MOHaM is also evaluated against the mobile, edge, and AR/VR workloads. Table 4.2 presents the workload scenarios considered along with their

TABLE 4.3: MOHaM 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	16		
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	168
Max. Shared Buf. Size	131 KiB	Max. PE Scratchpad Size	0.5 KiB
Simba-like Template			
Dataflow	Weight-Stat.	Max. Num. of PEs	128
Max. MACs per PE	32	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	256
Max. Neurons Buf. Size	131 KiB	Max. Synapses Buf. Size	131 KiB

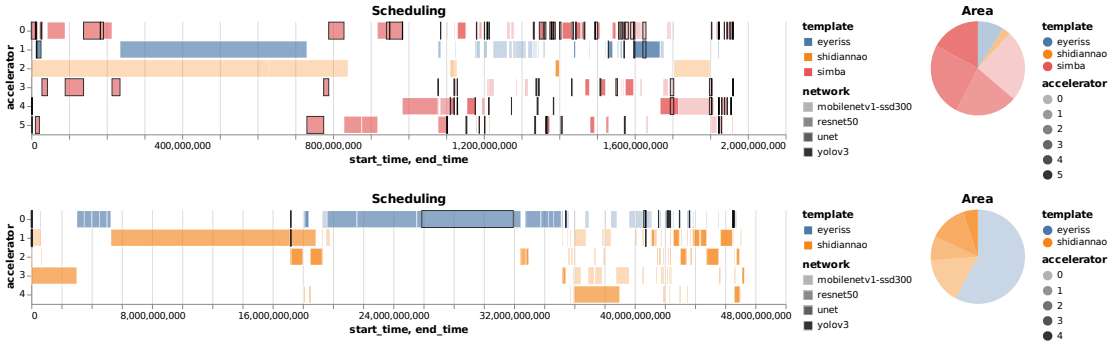


FIGURE 4.5: Comparison of scheduling Gantt chart and SA area contribution for two Pareto-optimal solutions

DNN models. They are provided to MOHaM as input in the ONNX [160] interoperable format. The *SAT* library used in MOHaM for the experiments is composed of the following state-of-the-art accelerators: Eyeriss [35] (Row-stationary dataflow), Simba [200] (Weight-stationary dataflow), ShiDianNao [57] (Output-stationary dataflow). In Table 4.3 we summarise the configuration of the MOHaM framework for the experiments in this section.

As described earlier, each configuration in the output Pareto-optimal set obtained using MOHaM consists of the MAS hardware description, the SAIs, their parameters and position in the NoP 2D mesh, and the scheduling of layers of the AM on the MAS. Figure 4.5 presents the scheduling Gantt chart and the area breakdown of SA for two Pareto-optimal solutions found by MOHaM for the AR/VR workload. The Gantt chart shows on the y-axis, the instantiated SAs (SAIs), and on the x-axis, the start and end times of the execution of each DNN layer, measured in cycles. Each bar represents the execution of a layer of the AM on a particular SAI. Layers from different DNN models are depicted with different opacity, while layers executed on instances of different SATs are depicted with different colours. Segments with black traces represent bandwidth-constrained execution segments as described in Section 4.2.3. The pie chart on the right displays the area contribution of each SAI with different opacity, namely the MAS area breakdown. Instances from the same SAT have the same colour. It can be noticed that the two solutions shown here are very different in terms of their scheduling and sub-accelerator instantiation preferences.

For all 3D plots shown in Figures 4.6, 4.7 and 4.8, the more left and top a solution lies, the better it performs. For the ease of readability, the plots report the projection of the solution points on three different planes, i.e., latency, energy and area. A common observation that is valid for all the results presented in this section is the distribution of the solutions found by the proposed MOHaM framework. In fact, they are spread over a large Pareto surface rather than being limited to a specific region. This is an important

advantage of MOHaM as it provides a variety of trade-off solutions from which the most appropriate for the specific use case can be selected.

4.3.1 Independent vs Simultaneous Optimisation

This experiment evaluates the need for hardware-mapping co-optimisation in multi-accelerator systems. Figure 4.6 shows the comparison of Pareto-optimal solutions with hardware-only, mapping-only and hardware-mapping co-optimisation. For hardware-only optimisation, the proposed MOHaM framework is run with only Simba-like SA templates to have a fixed dataflow (e.g., weight-stationary). For mapping-only optimisation, MOHaM is run with a fixed hardware configuration of 16 heterogeneous SAs, similar to MAGMA [110]. 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 lower energy and area but high latency. This is due to the fixed mapping (dataflow) strategy for all the layers. Whereas mapping-only optimisation (blue) has lower latency and energy but at the cost of a very high area. This is due to the fixed hardware configuration. In general, across all the workload scenarios, hardware-only optimisation has better energy and area but poor latency. Similarly, mapping-only optimisation has better latency and energy but high area. With hardware-mapping co-optimisation, the solutions (black) are very interesting. For example, for the *Data Center* workload, they have solutions with lower latency and energy along with minimum area. They have equally competitive solutions across other workloads. This is a result of the proposed MOHaM framework instantiating the SAs with optimal hardware resources and mapping the layers according to their dataflow preferences for optimal execution. Hardware-mapping co-optimisation can accommodate diverse workloads and offer the best overall performance in a multi-accelerator system.

4.3.2 Homogeneous vs Heterogeneous Accelerators

This experiment evaluates the need for heterogeneous SAs in multi-accelerator systems. Figure 4.7 shows the comparison of Pareto-optimal solutions with homogeneous and heterogeneous SAs. For homogeneous SAs, the proposed MOHaM framework 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 *Mobile* workload, Eyeriss (blue) has lower latency and area but at the cost of high energy. On the contrary, ShiDianNao (red) is both energy and area efficient but at the cost of very high latency. For the *Edge* workload, Simba (yellow) has the best while ShiDianNao has the worst latency, respectively. Eyeriss has lower latency, energy as well as area. In general, among the solutions with homogeneous SAs, Simba has better latency while ShiDianNao has a better area. With heterogeneous SAs, the solutions (black) are more uniformly distributed. For example, for the *AR/VR*

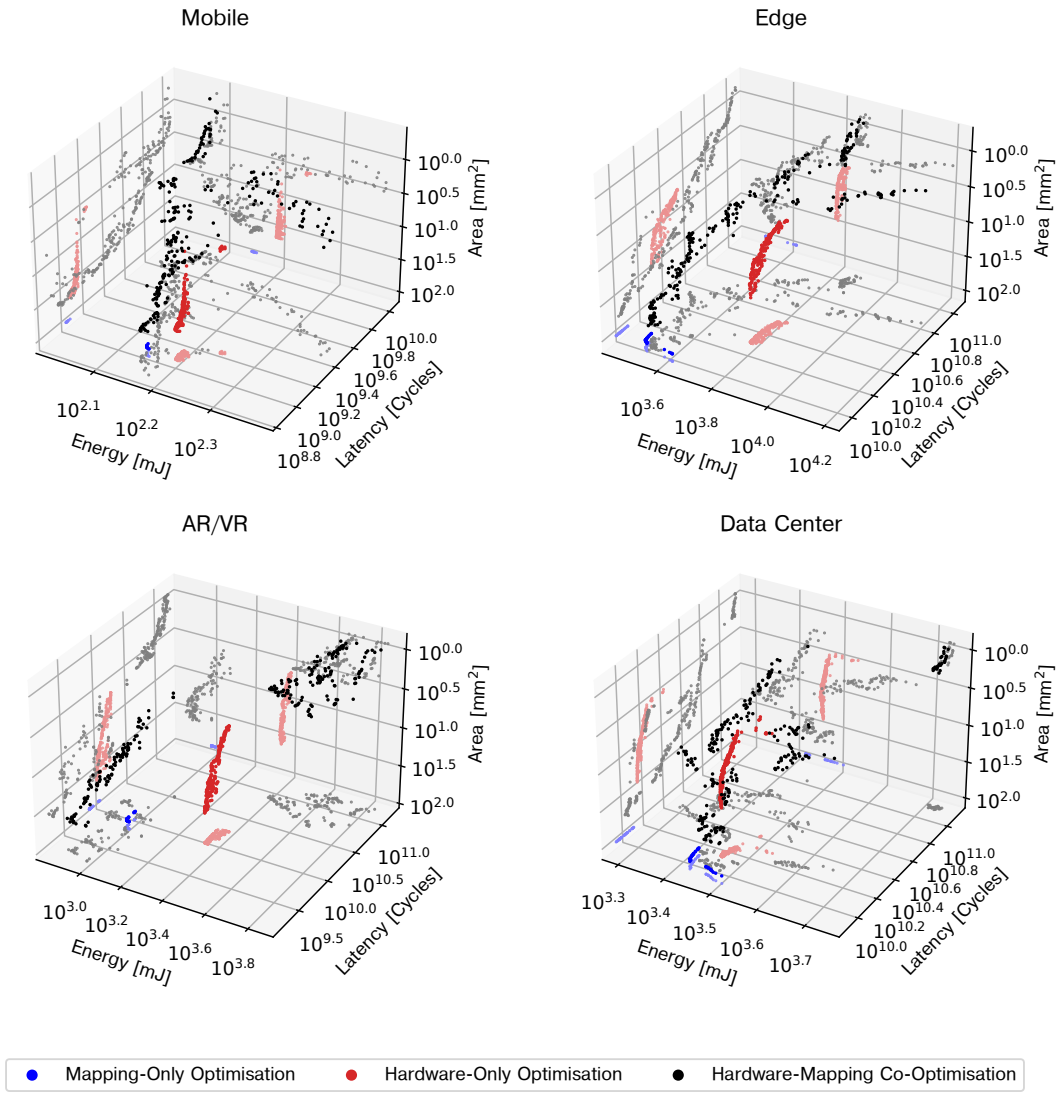


FIGURE 4.6: Comparison of Pareto-optimal solutions with hardware-only, mapping-only, and hardware-mapping co-optimisation

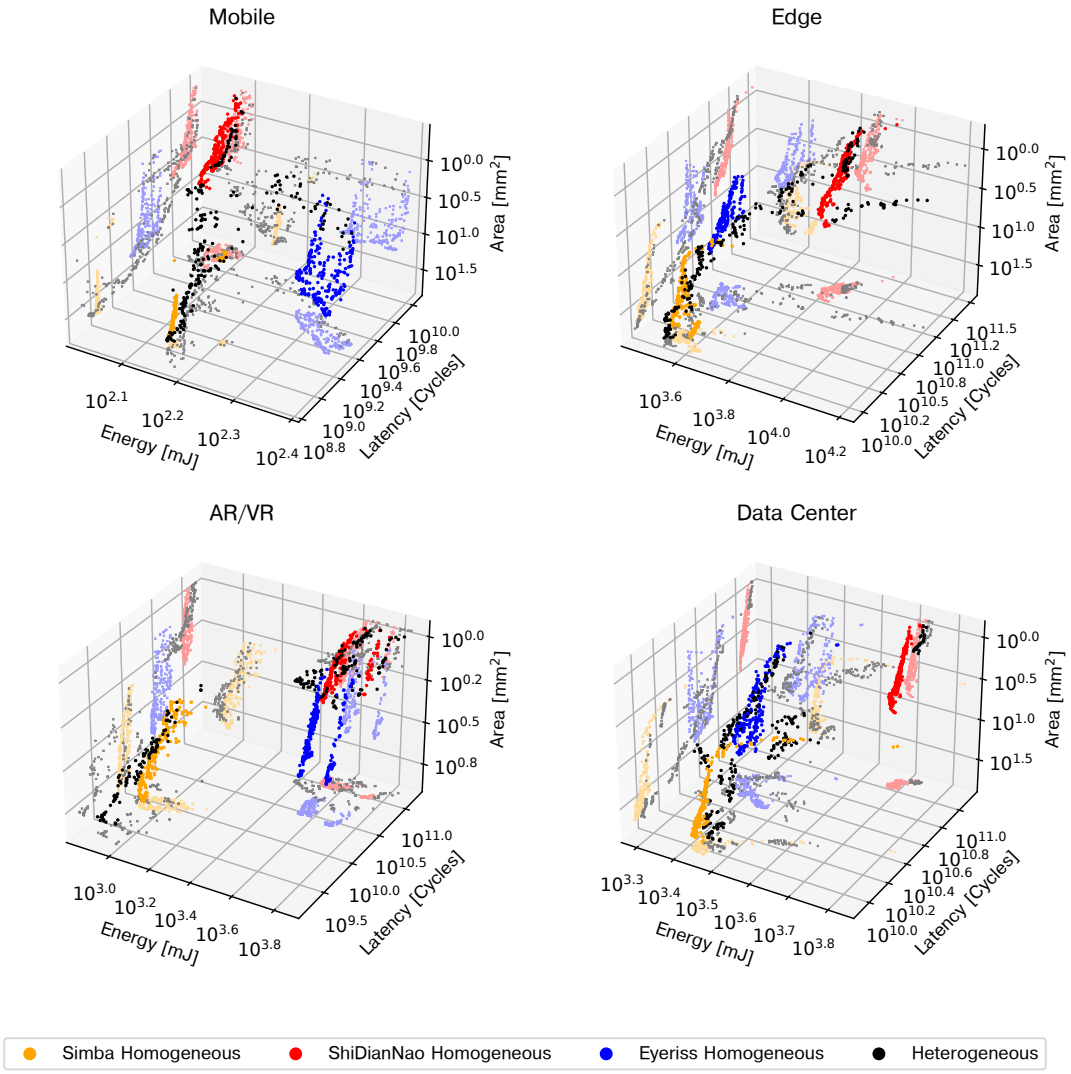


FIGURE 4.7: Comparison of Pareto-optimal solutions with homogenous and heterogenous sub-accelerators

workload, they have solutions with the lowest latency, lowest energy and lowest area. Their solutions are equally good for the *Data Center* and other workloads. It is possible as layers with specific dataflow preferences can be executed on the appropriate SAs. The proposed MOHaM framework instantiates the right set of SAs from the available templates and maps the layers for efficient execution. Hence, flexible dataflow with heterogeneous SAs increases the scalability of a multi-accelerator system toward diverse and emerging workloads.

4.3.3 Single-Objective vs Multi-Objective Exploration

This experiment evaluates the need for multi-objective exploration in multi-accelerator systems. Figure 4.8 shows the comparison of individual solutions with mono-objective and Pareto-optimal solutions with multi-objective exploration. For mono-objective exploration, the proposed MOHaM framework is run once, each with only latency, and energy as an objective. Then, they are compared with the result of a complete MOHaM run for multi-objective exploration. When objectives conflict, improving one results in worsening the other. For example, for the *Data Center* workload, the solution with energy as an objective (blue) is at the extreme left (i.e., best), but the latency and area are worst. Similarly, the solution with latency as an objective (red) is best while the area is worst. A multi-accelerator system design is usually explored with more than one objective. Existing works aggregate them into a single solution (e.g., EDP), which suffers if they conflict. Other commonly used forms of aggregations, like the weighted sum of cost functions, do not allow to explore the non-convex regions of the design space. MOHaM supports distinct multi-objective exploration and provides a Pareto-optimal set of solutions (black). For the same *Data Center* workload, it provides multiple competitive solutions with the lowest energy, latency and area. Similar solutions are available across all the workloads. Multi-objective exploration helps identify the most suitable design for a multi-accelerator system as per the need.

4.3.4 Comparison with state of the art

This section assesses the designs produced by MOHaM for multi-accelerator systems in comparison to three prominent approaches: CoSA [99], DiGamma [112], and Herald [128]. The publicly available implementations of CoSA and DiGamma are used, while Herald is implemented based on the specifications described in its evaluation methodology. CoSA focuses solely on mapping optimisation and employs Mixed-Integer Programming to generate a mapping strategy optimised with respect to the Timeloop [164] cost model. DiGamma takes a hardware-mapping co-optimisation approach, utilising a genetic algorithm in a manner similar to MOHaM, and relies on the MAESTRO [127] cost model for rapid design space exploration. Herald, meanwhile, uses exhaustive and

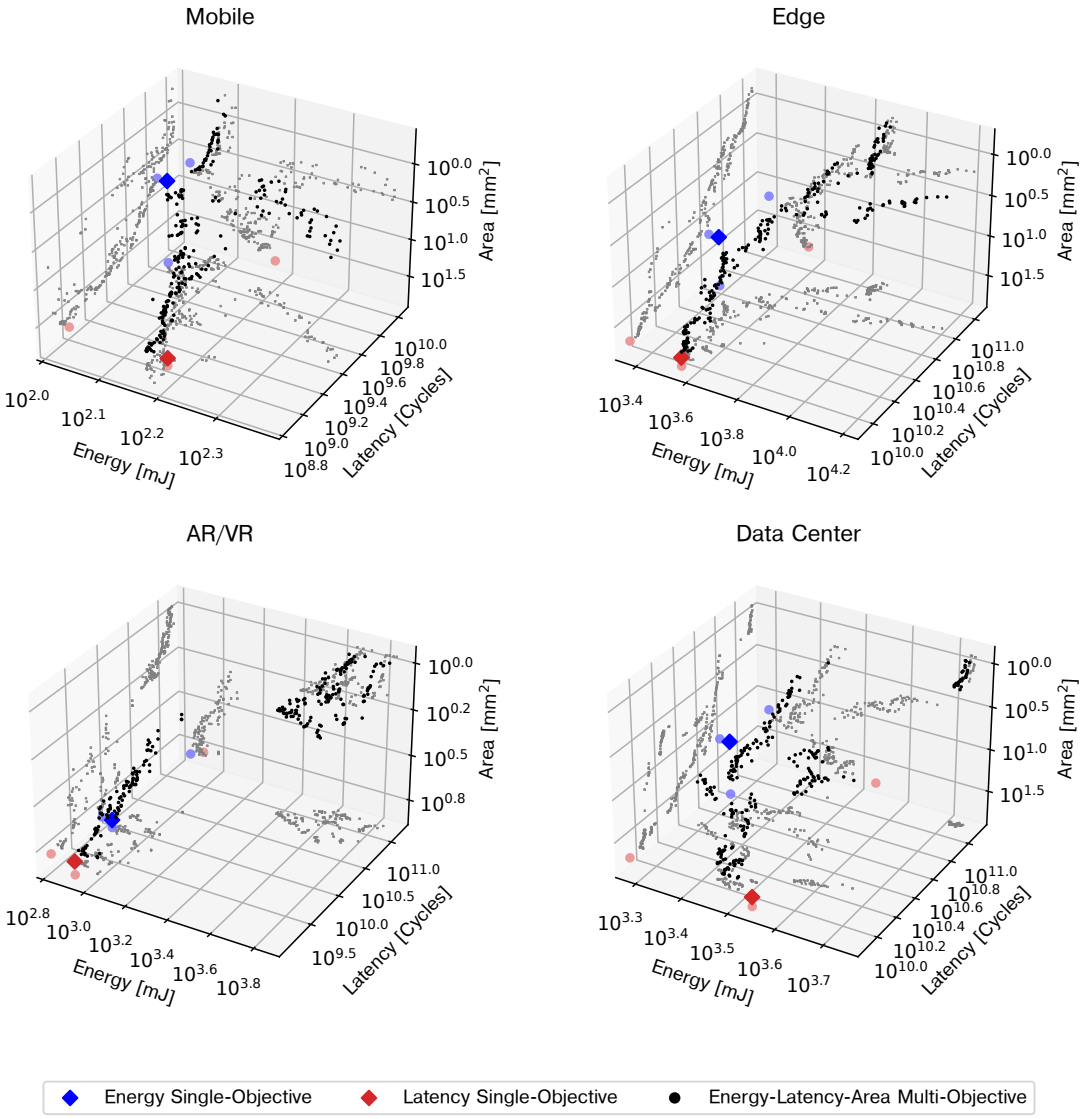


FIGURE 4.8: Comparison of individual solutions with mono-objective and Pareto-optimal solutions with multi-objective exploration

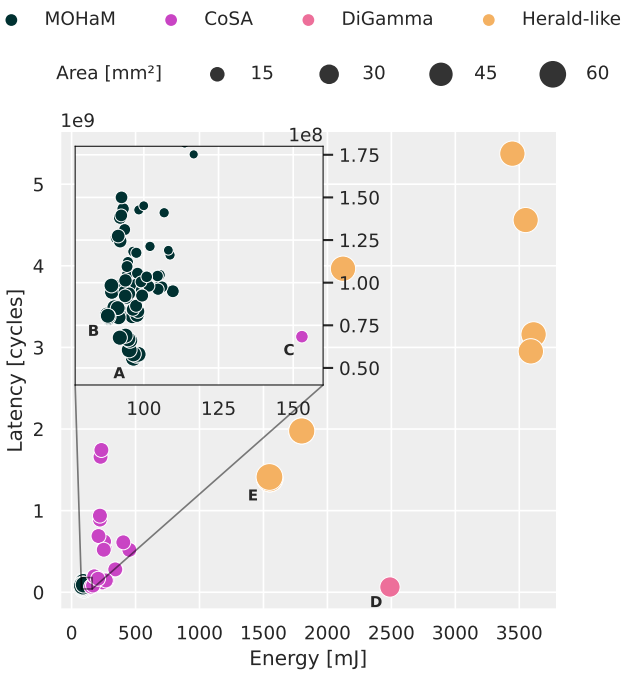


FIGURE 4.9: Comparison with state of the art

heuristic-based scheduling in a mono-objective hardware-mapping co-optimisation process for multi-accelerator systems, also leveraging the MAESTRO cost model.

Within the landscape of multi-accelerator system research, Herald distinguishes itself by targeting heterogeneous architectures, supporting workloads with multiple neural networks, and enabling hardware-mapping co-optimisation (see Table 4.1). Nonetheless, there are key differences between Herald and MOHaM. Herald focuses on distributing the MAC resource budget across a fixed set of heterogeneous accelerators, whereas MOHaM is capable of exploring both the optimal number and buffer sizes of accelerators. The mapping approach in Herald requires manual tuning, limiting flexibility and scalability to various accelerator templates. In contrast, MOHaM uses a genetic algorithm for mapping, which does not require manual adjustment for different templates or workloads. Moreover, Herald produces Pareto optimal solutions for single-objective design space exploration, while MOHaM provides Pareto optimal solutions for scenarios with multiple objectives.

Herald was originally evaluated using the MAESTRO cost model, but for a fair comparison with MOHaM, the Timeloop cost model is adopted here. This variant is referred to as *Herald-like* in this study. In addition, Herald-like is configured with the same number of MACs and accelerator instances as the maximum configuration used for

TABLE 4.4: Configuration of design points from Figure 4.9

	Design Points				
Metrics	**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

MOHaM. The simulations for CoSA and DiGamma are conducted on architectures that allow flexible dataflow, without considering the overheads of reconfiguration. For CoSA, an architecture similar to Simba is assumed, with the number of MACs matching the maximum MOHaM configuration. CoSA's objective function combines compute, traffic, and buffer utilisation as a linear combination, and multiple solutions are obtained by varying the weighting of these components. While DiGamma is typically evaluated with MAESTRO, an implementation with Timeloop is available and used here. As DiGamma operates at the layer level, results are aggregated across all layers, and the area constraint is set to match MOHaM's maximum configuration.

Figure 4.9 presents a selection of Pareto design points (exhibiting interesting trade offs) generated by MOHaM, alongside those produced by CoSA, DiGamma, and the best-performing subset of Herald-like, for the AR/VR workload. Table 4.4 details the configurations of the design points (labeled A through E) used in the comparison. The ability of MOHaM to produce Pareto-optimal design points provides competitive alternatives to CoSA and DiGamma. For instance, MOHaM's design point A achieves an 18.40% reduction in latency and a 36.87% reduction in energy compared to CoSA's design point C. Despite having more MACs, CoSA's latency remains higher, which can be attributed to the absence of hardware optimisation and lack of awareness of memory bandwidth limitations. The execution in CoSA is found to be memory-bound, as shown in Table 4.4. The greater number of MACs, and consequently processing elements, leads

to increased communication overhead and higher energy consumption. Since CoSA does not include hardware optimisation, its area is set to the minimum MOHaM configuration for a fair comparison, resulting in design point C having the smallest area among all compared points.

A similar trend is observed when comparing MOHaM's design point A to DiGamma's design point D. MOHaM achieves a 13% reduction in latency and a 96.12% reduction in energy. Although DiGamma has the same number of MACs as CoSA and supports hardware-mapping co-optimisation, its latency is slightly higher. This is attributed to its lack of architectural template reconfigurability: DiGamma supports only a two-level hierarchy with processing element and shared buffers, whereas MOHaM supports multi-level hierarchy with all architectural templates compatible with the Timeloop cost model. DiGamma's output logs do not provide insight into potential memory bottlenecks. The elevated energy consumption in DiGamma is due to its large processing element buffers, which occupy 71.85% of the total area and lead to frequent high-energy memory accesses.

For Herald-like, design point E represents the best energy-delay product solution. Nevertheless, MOHaM's design point A reduces latency by 96% and energy by 93.76% compared to E. Even with hardware-mapping co-optimisation and layer scheduling aimed at balancing load, Herald-like exhibits higher latency and energy usage. This is a consequence of the random nature of its hardware and mapping search, coupled with the absence of multi-objective optimisation. In an attempt to improve latency, Herald-like allocates large processing element and shared buffers, which results in only 5.68% of its area being dedicated to computation (see Table 4.4). As a result, execution becomes compute-bound, increasing latency. Large buffers also increase the number of high-energy memory operations, raising overall energy consumption. Additionally, MOHaM's design point A achieves a 39.78% area saving compared to Herald-like's design point E.

For designers of multi-accelerator systems, the availability of multiple competitive design points is advantageous, especially when multiple objectives are considered. Mono-objective optimisation, by contrast, collapses all criteria into a single design point, which is only justifiable in the absence of conflicting objectives—a rare situation. For example, when Herald's mono-objective approach aggregates latency and energy into a single metric, neither is reduced effectively. The strategy of allocating large buffers to reduce latency inadvertently increases it, and such large buffers are not helpful for energy reduction. This scenario results in high energy consumption due to conflicting objectives. In contrast, multi-objective optimisation allows for the identification of several competitive design points that address all objectives without disproportionately penalising any. MOHaM's approach, for instance, enables the selection of a solution such as design point A, which finds a balance between latency, energy, and area. Depending on requirements, other design points may also be preferable; for example, MOHaM's design point B achieves notable improvements in energy efficiency over CoSA, DiGamma, and Herald-like, with trade-offs in latency and area.

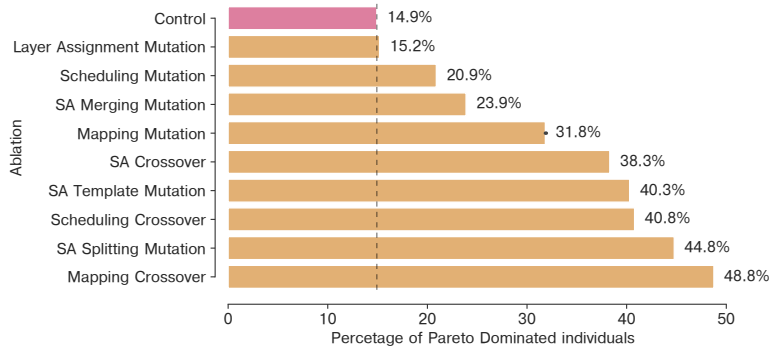


FIGURE 4.10: Percentage of Pareto-dominated solutions when an operator is ablated from the baseline MOHaM configuration

4.3.5 Ablation Study

This experiment evaluates the effectiveness of custom genetic operators implemented for the MOHaM framework. It compares the result of a complete MOHaM run with the results of runs, each with one operator disabled (ablated). Figure 4.8 shows the percentage of Pareto-dominated solutions when an operator is ablated from the baseline MOHaM configuration (refer Table 4.3). Due to the multi-objective exploration, the results are obtained as follows: *(a)* MOHaM is run with the default configuration, and a baseline Pareto-optimal set of individuals is selected from the final population. *(b)* MOHaM is re-run with the same configuration to get a second Pareto-optimal 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 4.10. *Control* serves as the threshold to compare ablation results. *(d)* MOHaM is run after disabling one custom genetic operator to get a new Pareto-optimal set of individuals. *(e)* Steps *(c)* and *(d)* are repeated for all the operators and the results are presented in Figure 4.10. A higher percentage of Pareto-dominated individuals indicate that the operator is more effective and that the performance of MOHaM will deteriorate without it. All the operators perform better than the *Control* threshold, implying that each one of them has some significance in the MOHaM framework.

4.3.5.1 Sampling Efficiency

This experiment evaluates the sampling efficiency of the GA-based MOHaM algorithm in comparison to a random exhaustive search. A set of specialized genetic operators is crafted to enable a more rapid and sample-efficient DSE using MOHaM. Figure 4.11

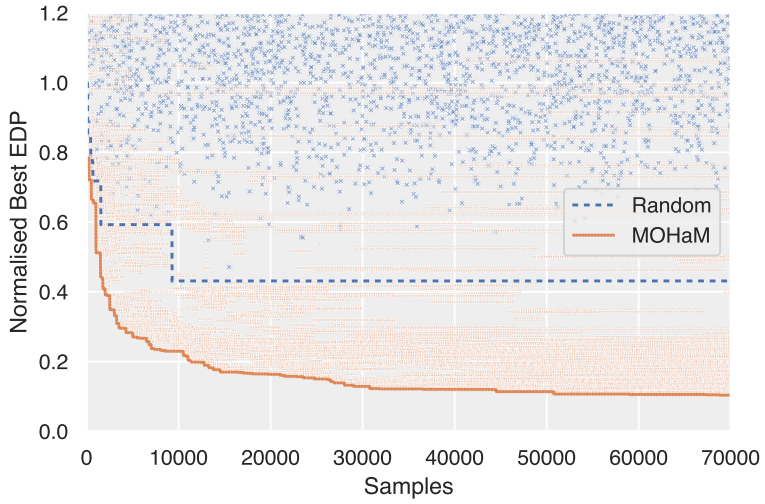


FIGURE 4.11: Sampling efficiency of MOHaM algorithm.

illustrates the normalized Energy-Delay Product (EDP) for 70,000 solution samples obtained from both a random search and a MOHaM-based DSE. To illustrate the evolution of solution discovery, a curve traces the highest EDP samples. Upon completion of these 70,000 samples, the MOHaM achieves an EDP improvement of 4.17× over the random search. It is noted that MOHaM continues to evaluate some solutions with high EDP due to its inherent multi-objective exploration capability. Whereas solutions from random searches are widely dispersed and largely exhibit poor EDP. The solution curve distinctly indicates that a DSE implemented with MOHaM is significantly faster, by orders of magnitude, than a random exhaustive search.

4.4 Conclusion

In this chapter, we introduced MOHaM, a comprehensive framework for multi-objective hardware-mapping co-optimisation tailored to multi-DNN workloads on chiplet-based accelerators. The principal findings of this work are as follows: (1) employing flexible dataflows with heterogeneous sub-accelerators (SAs) enhances the scalability of multi-accelerator systems, enabling them to effectively accommodate diverse and emerging workloads; (2) hardware-mapping co-optimisation supports a wide range of workloads while achieving optimal overall performance in multi-accelerator systems; (3) leveraging multi-objective exploration facilitates the identification of the most suitable designs for specific system requirements; (4) the proposed MOHaM-based DSE significantly accelerates the search process, outperforming random exhaustive search by several orders of magnitude; and (5) substituting Timeloop with a faster cost model can further reduce

the execution time of the MOHaM framework. Potential applications of MOHaM include the design of DNN-powered Advanced Driver-Assistance Systems (ADAS), such as those deployed in Tesla vehicles [213], and DNN-based AR/VR hardware for the Metaverse [146], both of which feature well-defined workloads. Future directions for this research involve advancing the exploration of the scheduling space to improve chiplet (SAIs) utilisation.

Chapter 5

Online Scheduling in DNN Multi-Tenant Systems via RL

For service providers, the management of multi-tenancy and the assurance of high-quality service delivery, especially with regard to meeting strict execution time constraints, are of critical importance, all while striving to maintain cost-effectiveness. In this context, the adoption of heterogeneous multi-accelerator systems is increasingly pertinent. This chapter introduces RELMAS, a low-overhead deep reinforcement learning algorithm tailored for the online scheduling of deep neural networks (DNNs) in multi-tenant environments, explicitly addressing the challenges of accelerator dataflow heterogeneity and memory bandwidth contention. By employing RELMAS, service providers are empowered to implement the most efficient scheduling policies for user requests, thereby optimizing Service-Level Agreement (SLA) satisfaction rates and improving overall hardware utilization. Empirical evaluation of RELMAS on a heterogeneous multi-accelerator system comprising diverse instances of Simba and Eyeriss sub-accelerators, demonstrates up to a 173% enhancement in SLA satisfaction rates compared to state-of-the-art scheduling approaches across a variety of workload scenarios, incurring less than 1.5% energy overhead.

The research presented in this chapter has been presented at *61st ACM/IEEE Design Automation Conference* (DAC 2024) held in San Francisco and at the *2024 IEEE International Symposium on Circuits and Systems* (ISCAS 2024) held in Singapore and published in the proceedings [22, 187].

5.1 Introduction

Executing DNNs is neither straightforward nor cost-effective, primarily due to their substantial hardware requirements. With the continual development of new DL models, which increasingly comprise larger numbers of parameters, the need for greater memory and computational resources is ever-growing [231]. While computational requirements escalate, computational power is not advancing at the same pace. As advocated in

previous chapters, a prevalent solution is the adoption of DSAs for DNN execution. Simultaneously, applications that require fast, accessible, and cost-effective solutions are progressively offloading DNN execution to cloud services, which leverage the computational capabilities of large accelerators in data centers. In engineering, there is a trend towards simplifying complex processes, making advanced functionalities accessible to users and developers without deep technical expertise. This is evident in paradigms such as Function-as-a-Service (FaaS, i.e., Serverless Computing) [14], which has been extended to Machine Learning tasks through the Inference-as-a-Service (INFaaS) paradigm [185]. In this way, software developers can delegate computation-heavy tasks to cloud services, allowing them to focus on their core application logic. As a result, effective management of hardware resources becomes a critical concern for service providers.

A promising short-term solution is the development of scheduling algorithms that efficiently allocate different neural networks from multiple tenants onto shared hardware resources, typically organized as multi-accelerator systems (MAS) composed of multiple sub-accelerators (SAs). These algorithms must guarantee user-defined constraints, particularly ensuring that instances of DNN models are executed within specified maximum deadlines.

In this chapter, we present RELMAS, an approach designed to tackle the challenge of concurrently scheduling multiple DNN models on a MAS with heterogeneous SAs. The primary goal is to ensure various levels of Quality of Service (QoS), especially concerning the satisfaction of maximum deadline constraints. RELMAS is an online scheduling method that models the environment as a MAS with a fixed number of SAs and a dynamic set of DNN model instances that arrive online and must be scheduled and completed within their respective deadlines. Our proposed solution utilizes Deep Reinforcement Learning, specifically the Deep Deterministic Policy Gradient (DDPG) algorithm. Unconventionally, we coordinate the DDPG algorithm with Long Short-Term Memory (LSTM) networks, providing a mechanism to automatically discern deadline constraints among different DNN model instances and generate schedules that effectively balance workloads across both space and time.

The algorithm demonstrates effective convergence, yielding a substantial reduction in deadline misses and a significant improvement in deadline compliance.

5.2 Related Work

While the topic of multi-tenancy DNN scheduling in conventional general-purpose cores has been widely studied in both software and hardware since the introduction of chip-multi-processors, addressing multi-tenancy execution in DSAs is a more recent endeavor. Diverse approaches have emerged from academic research that can be classified into two categories [118]: static and dynamic. Static mechanisms involve configuring software or hardware statically (i.e., no decisions are taken at runtime) to adjust the memory

access rates of contentious applications. Dynamic mechanisms, on the other hand, utilize runtime information to adaptively adjust the contentious nature of an application based on the actual amount of memory traffic in the system.

Prior works like LayerWeaver [159], PREMA [40], and AI-MT [11] propose time-multiplexing DNN execution, either statically or dynamically. However, they often suffer from low hardware utilization, especially when layers cannot fully utilize all available resources. To enhance hardware utilization, static partition techniques such as HDA [128] and MAGMA [110] propose spatial partitioning of compute or memory resources. However, their static nature hinders adaptability to different dynamic scenarios. While AI-MT employs multi-accelerators with homogeneous cores (specifically, systolic array architecture), MAGMA also supports multi-accelerators with heterogeneous cores. Additionally, the multi-tenancy support in AI-MT relies on heuristics, while MAGMA uses a research process based on a genetic algorithm. AI-MT considers potential inter-layer dependencies, unlike MAGMA. MAGMA is an offline scheduling strategy, requiring all job characteristics and arrival times to be known beforehand, which does not align with the requirements of on-demand services. Recently, dynamic spatial partition mechanisms have been suggested to further improve co-location efficiency. Planaria [73] and Veltair [140] propose dynamic allocation of compute resources for co-running workloads. Despite their ability to adaptively allocate compute resources, such as processing elements for Planaria or CPU cores for Veltair, they lack the capability to dynamically manage memory resources. This leads to low resource utilization and high thread migration overhead when repartitioning compute resources. In contrast, MoCA [118] highlights the importance of memory-centric resource management, proposing a dynamic partition of both compute and memory resources throughout execution.

Although homogeneous-core accelerators have been thoroughly explored in the literature, heterogeneous architectures, often incorporating varied dataflows, enable more efficient processing of a diverse range of DNN layers, serving as both flexible and cost-effective solutions [209]. However, in a data center environment that provides on-demand inference services – with unpredictable job arrivals and Service Level Agreements (SLAs) primarily focused on execution within set deadlines – the necessity arises for RELMAS, a sample-efficient, online DNN scheduling algorithm, that mitigates the risk of resource underutilization and QoS violations while maintaining bandwidth constraints.

5.3 Problem formulation

We consider a fixed hardware accelerator architecture, which consists of M heterogeneous SAs, such as the one depicted in Fig.5.1. In the context of this MAS, individual SAs exhibit distinct specifications encompassing dataflow capabilities, processing unit counts, memory hierarchy configurations, and buffer sizes. All the SAs share the off-chip memory bandwidth. Such a system, which can ideally be deployed in both data center and edge

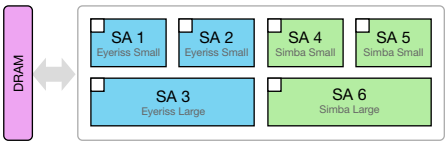


FIGURE 5.1: Diagram of the reference Multi-Accelerator Heterogeneous Architecture used in evaluations.

scenarios, operates in a multi-tenant fashion, meaning it can receive and fulfill inference requests from multiple users. The resulting workload within the MAS can be parallelized at the layer granularity, allowing for concurrent processing of different layers on different SAs.

A user request comprises the task of executing a DNN model inference while adhering to latency constraints specified in a SLA. We refer to each request as a job. Each job consists of several sub-jobs (SJs): one for each layer of the requested DNN model. At runtime, all the sub-jobs to be executed are collected in a virtually infinite ready queue (RQ). SJs are added to the RQ when a new request arrives and are removed when they start their execution on a SA or their deadline (stemming from the SLA) passes.

The objective of this work is to design a scheduling algorithm with the primary goal of maximizing the SLA satisfaction rate, namely the fraction of jobs completed while meeting the specified latency requirements, or conversely minimizing the deadline misses.

We make the reasonable assumption that all potential DNN models that may be requested are known in advance. This is in line with prior works [205]. Layer dependencies and parameters of all these models are known. Thus, it is possible to compile a table containing: the latencies $c_{i,s}^{m}$ and the required bandwidths $b_{i,s}^{m}$ to execute the s-th SJ of the i-th potential request on the m-th SA. Due to the heterogeneity of the MAS, each layer executed on different SAs exhibits varying latency, energy, and bandwidth requirements. However, in an online context, during periods of low system activity, any unfamiliar model could undergo a "registration phase", involving compiling latency tables for the new model. Once registered, these models are integrated into the system.

On the other hand, arrival times of each request a_j are not known in advance. Thus, an online scheduling strategy is needed. Such a scheduler should determine the start time s_j for each SJ and the SA on which the SJ will be executed. A deadline miss will happen if the finish time $f_j > s_j + q_j$, where q_j is the latency requirement of the request. The finish time cannot be simply calculated summing the computational costs of each layer on the assigned SA because of the limited memory bandwidth shared by all the SAs. When the sum of the bandwidth required by multiple SJs executing in parallel on multiple SAs exceeds the available memory bandwidth, all the SJs experience a slowdown proportional to the respective required bandwidth. This translates to the same amount of stall cycles for all the overlapping sub-jobs. Hence, it is important for the online

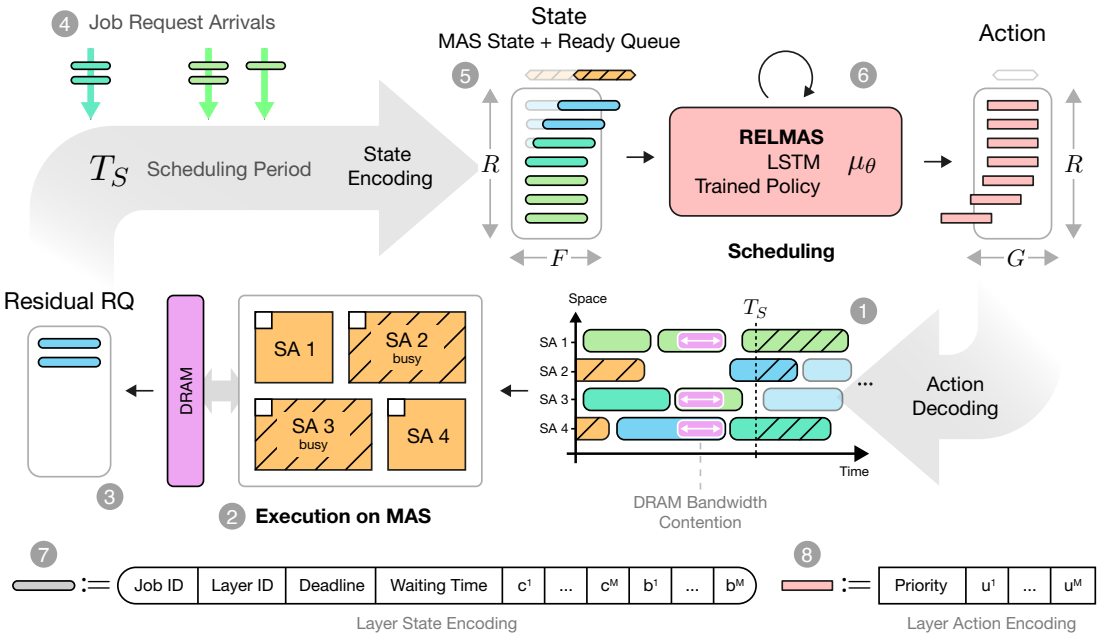


FIGURE 5.2: High-level visualization of the RELMAS policy in action during deployment.

scheduler to wisely orchestrate the execution of compute-intensive and memory-bound sub-jobs in the RQ to minimize bandwidth contentions.

The described problem can be considered as a generalization of the Flexible Job Shop Scheduling (FJSP) problem and, in Graham's notation [81], it can be expressed as $FJSP \mid prec \mid U$, where $prec$ refers to the presence of precedence constraints among the SJs and U is the number of tardy jobs, which is the objective to be minimized. It has been proved that the FJSP problem is NP-hard [50].

5.4 RELMAS Online Scheduler

In this section we introduce RELMAS, the proposed approach to online scheduling in a multi-accelerator heterogeneous system based on Deep Reinforcement Learning (DRL). For this section, we adopt a top-down approach: we first depict the working of RELMAS in a deployment setting, then we deepen the RELMAS agent learning process. For an introduction to the fundamentals of deep reinforcement learning, please refer to [8].

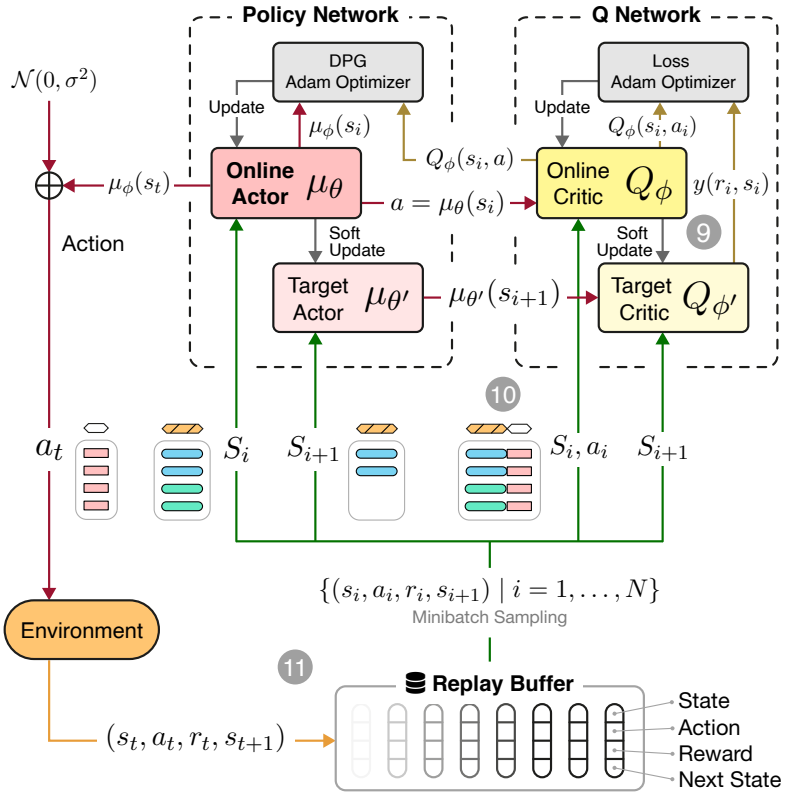


FIGURE 5.3: RELMAS policy learning process.

5.4.1 Scheduling

In the target scenario, the environment consists in the MAS executing DNN layer inferences and receiving new requests. The latter represent the stochastic component of the environment. The state of the environment at any given time includes the current system load information, i.e., the number of cycles for which each SA will be busy executing a previously assigned SJ, and the RQ, i.e., the collection of SJs ready to be executed. The action, which is the decision taken by the RELMAS policy agent for each SJ, can be articulated in a priority, defining temporal scheduling, and a SA choice, defining spatial allocation of the SJ in the MAS. Fig. 5.2 shows an overview of RELMAS in a deployment setting. It can be noted that the scheduler is triggered periodically with a period T_s. During a period, the MAS execute a previously elaborated scheduling, which is represented graphically in the space-time graph in Fig. 5.2.1. At the end of the period, SJs remained unexecuted are collected in the so called residual RQ (Fig. 5.2.3). These SJs are merged with the new SJs related to job requests arrived and queued in

the meantime (Fig. 5.2.4).

This overall RQ (Fig. 5.2.5) is sorted in ascending order of absolute deadline and submitted for scheduling by the RELMAS policy. We decided to implement the RL policy as a Long-Short-Term-Memory (LSTM) recurrent neural architecture. The rationale is that it helps dealing with a state of variable size (variable number of SJs in RQ) and LSTMs are good in detecting temporal relationships and patterns of SJs sequences. The LSTM cell with a hidden size of h is succeeded by two fully connected layers: the first projects to $h/2$ units and applies a Rectified Linear Unit (ReLU) activation function, while the second projects to G values and applies hyperbolic tangent (tanh) activation function, encoding the decision, i.e., the action, for the corresponding SJ.

As shown in Fig. 5.2.6, the LSTM sequentially ingests the encoded representation of each SJ in the RQ. Hence, if R sub-jobs reside in the RQ, R timesteps of the LSTM policy are executed. The encoded representation (Fig. 5.2.7) of each SJ is an array of features including all the information that could be useful to derive an optimal scheduling: an identifier of the model requested of which the SJ represent a layer, an identifier of the layer, the deadline representing the latency constraint[1], the waiting time informing about the distance in time from the arrival time of the request, the computational times and bandwidth requirements on the SAs. Thus, the length F of each encoding is equal to $4 + 2M$.

In Fig. 5.2.5 it is also possible to spot an extra element prepended to the encoded representation of the RQ. As no preemption is assumed, it is important to instruct the policy about the duration for which each SA will be busy. We do this by using a virtual SJ as a primer for the LSTM. All its features are zero except for computational times that instead represent the number of cycles before each SA will be available.

As output of the LSTM policy, after discarding the first timestep (corresponding to the primer) we get a decision encoding (Fig. 5.2.8) for each SJ in the ready queue. Due to the Tanh activation function, all the output values relies in the $[-1, 1]$ range that for each SJ are: the temporal priority and one value u_m for each SA. The index associated with the highest u_m determines the spatial allocation of the SJ in the MAS. Thus, the length G of each encoding is equal to $1 + M$.

5.4.2 Learning

As shown in Fig. 5.3, we employ Deep Deterministic Policy Gradient (DDPG) which has emerged as a prominent algorithm in the field of reinforcement learning, particularly for solving high-dimensional continuous action space problems. For the sake of brevity, we will not describe all the mathematical details of DDPG, please refer to [137] for a

[1]The deadline for each job request is explicitly stated and reiterated within the representation of each SJ. This redundancy enables the policy to differentiate between two identical SJs (same layer, same model), each associated with distinct request latency constraints.

detailed explanation. In this section, we instead focus on the choices made to adapt DDPG to the problem addressed by this work.

As in other RL algorithms, the goal in DDPG is to learn an optimal policy that maximizes a cumulative reward signal over time. To accomplish this, DDPG employs two essential components: the actor and the critic networks. Both are aimed to construct parameterized functions that iteratively converge to the corresponding optimal functions. Due to the exceptional capability of DNNs to approximate non-linear functions, two neural networks are used for these components. The actor network represent the parameterized deterministic policy function μ_θ and provides a continuous action output given an input state representation. On the other hand, the critic network estimates the Q-function Q_ϕ, which quantifies the expected cumulative reward of following a given policy for a specific state and action pair.

We already discussed the policy as an LSTM-based architecture. Likewise, we propose to adopt the same architecture for the critic network (Fig. 5.2.9). The state and action pairs are concatenated horizontally as shown in Fig. 5.2.10 and provided as input to the critic for Q-value estimation. Thus, the input length at each timestep is equal to $F + G$, while only one output value $Q_\phi(s, a)$ per timestep is projected from the last hidden state.

As in the baseline DDPG algorithm, experiences are stored in a *replay buffer* (Fig. 5.2.11) and sampled from it during the training phase for weight updates. Each experience is a tuple (s_t, a_t, r_t, s_{t+1}): s_t is the environment state encoding at time t, a_t is action representation of the decision taken by the policy, r_t is the reward value obtained as an effect and s_{t+1} is the state representation after executing the action. As we assume RELMAS to work in a periodic fashion and due to the stochastic request arrivals, s_{t+1} can also include stochastic request arrivals, therefore breaking a deterministic causality chain from s_t to s_{t+1}. Indeed, we found out that learning performance increases when s_{t+1} only encodes the residual RQ instead of the overall RQ, eliminating the stochastic component from the experiences and allowing the algorithm to effectively infer causality chains.

Regarding the reward function, we defined it as follows:

$$r_t = \sum_{l \in RQ_t} \Delta_{t,l} \left(A_{t,l} + \gamma \frac{(a_l + q_l) - f_l}{q_l} \right) \tag{5.1}$$

where

$$\Delta_{t,l} = \begin{cases} 1, & f_l < t + T_s \\ \delta, & \text{otherwise} \end{cases} \tag{5.2}$$

$$A_{t,l} = \begin{cases} \alpha, & f_l \leq a_l + q_l \\ -\beta, & \text{otherwise} \end{cases} \tag{5.3}$$

and $\delta \in [0, 1]$ quantifies the relevance of events beyond the next scheduling period, α is the deadline hit reward, β is the deadline miss penalty and γ weights the significance of SJs normalized slacks. The introduction of normalized slacks, implemented as a scaled sum term, effectively mitigates resource underutilization. While compliance with SLA requirements is the foremost priority, this strategy additionally incentivizes the reduction of resource usage duration. This alignment serves the dual objective of not only fulfilling service commitments but also boosting operational efficiency.

5.5 Evaluation

In this section, we present the evaluation of our novel online scheduling technique. To conduct our evaluation, we considered the hardware specifications detailed in Table 5.1. Our chosen hardware configuration is rooted in a heterogeneous accelerator architecture following the principles described in [128], boasting an equivalent off-chip memory bandwidth and total number of MAC units as in other works focusing on systolic arrays [118]. We are aiming for a multi-chip-module-based architecture in which sub-accelerators take the form of chiplets interconnected by means of a network-on-package (NoP). The on-package links utilize a silicon interposer and incorporate efficient intra-package signaling circuits, as in [200]. The selected architectural configuration comprises six sub-accelerators, which are instances of two widely recognized state-of-the-art DNN accelerators, as visualized in Fig. 5.1. Specifically, half of these sub-accelerators adopt a row-stationary dataflow and an internal organization closely resembling Eyeriss [36], while the remaining half employ a weight-stationary dataflow and an architecture akin to a Simba [200] chiplet. We have chosen to integrate both small and large instances from these two categories of sub-accelerators (see Table 5.1). This strategic decision enables more effective orchestration of memory-bound and compute-intensive workloads.

In our evaluations we consider seven different state-of-the-art DNN models from image classification, object detection and natural language processing (NLP) domains. These DNN models features different model sizes, computational, memory and bandwidths requirements rendering them highly representative of a wide spectrum of DNN workloads. Similarly to [73] and [118], we organized these models in three workload sets classified by size as summarized in Table 5.2. To generate multi-tenant scenarios from each of these three workload sets, we consider a time interval during which inter-arrival times of inference requests are drawn from a Pareto distribution, emulating task dispatching in data centers according to insights from [45].

Each request is associated with a SLA requirement defined by the QoS level. Adopting the PREMA [40] approach, the latency requirement is calculated multiplying a coefficient, that we name QoS factor, by the minimum execution latency of the requested job (i.e., without interference of other jobs). Specifically, we identify a QoS factor for the baseline QoS level (QoS-Medium), then we derive other two targets: QoS-Low which

TABLE 5.1: Parameters of the sub-accelerators classes considered in evaluations.

	Eyeriss		**Simba**	
Parameter	**Small**	**Large**	**Small**	**Large**
Frequency	1 GHz			
DRAM Bandwidth	16 GB/s			
NoP Bandwidth, Energy	100 GB/s, 1.3 pJ/bit			
Dataflow	Row Stationary		Weight Stationary	
Num. PEs	256	512	16	32
MACs per PE	1	1	16	16
Global Buffer	64 KiB	64 KiB	32 KiB	64 KiB
PE Buffers	220 B	220 B	24 KiB	24 KiB

TABLE 5.2: Benchmark workloads used in evaluations.

Workload	**Domain**	**DNN Models**
Light	Image Classification	SqueezeNet
	Object Detection	YOLO-Lite
	NLP	Keyword Spotting
Heavy	Image Classification	AlexNet, InceptionV3, ResNet50
	Object Detection	YOLO-V2
Mixed	Light ∪ Heavy	

indicates relaxed latency constraints corresponding to 1.2× QoS-Medium and QoS-High denoting 0.8× QoS-Medium, similarly to [73] and [118].

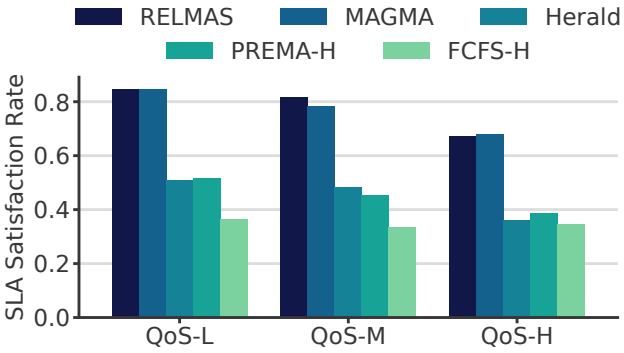
We assess the energy, latency and memory bandwidth requirements figures of each DNN layers on each sub-accelerator using the validated Timeloop/Accelergy [164] simulation infrastructure. Additionally, we develop a multi-accelerator multi-tenant simulation platform that takes into account layer latencies, layer dependencies and runtime memory bandwidth contentions to accurately determine the start and finish times of each sub-job when executing according to a specific scheduling algorithm. This methodology allows us to effectively evaluate the target metrics. Across all the evaluations, the following reward coefficients were used during RELMAS learning: α = 0.10, β = 0.11, γ = 0.05 and δ = 0.01.

5.5.1 SLA Satisfaction Rate

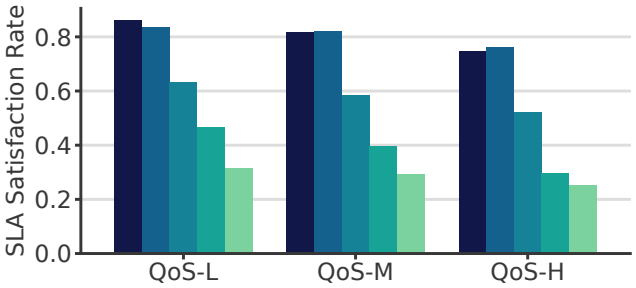
In Fig. 5.4 we show the results of the evaluation of the three workload sets in Table 5.2 with three different QoS targets. We adopt the SLA Satisfaction Rate as our primary comparative metric. To evaluate the efficacy of the proposed approach, we conduct comparative analyses against robust baselines, which draw inspiration from and are adapted from previous works in the field. The following baseline algorithms are considered: (1) FCFS-H, which is the First-Come-First-Served scheduling algorithm for priority assignment to each sub-job in the ready queue, paired with an heuristic for the selection of the sub-accelerator. The heuristic consists in choosing the SA which minimizes the finish time of the sub-job. (2) PREMA-H, which is a scheduling algorithm based on [40], combining a token mechanism based on waiting time and Shortest-Job-First priority assignment. As the original PREMA work targets a monolithic architecture, for a fair comparison we combine it with the aforementioned heuristic. (3) Herald [128], which is a layer scheduling algorithm designed for heterogenous architectures, focusing on load balancing of the SAs. (4) MAGMA [110], consisting in a genetic algorithm approach to scheduling optimization. Our implementation adopts the same custom genetic operators of the original work. For a fair comparison, we used the same evaluation platform as RELMAS and adapted its fitness function to take into account the SLA satisfaction rate.

All the evaluated algorithms are supposed to be triggered for each scheduling period at runtime. For each tested scenario we trained RELMAS considering a hidden size of 256 for both the actor and the critic. As shown in Fig. 5.4, the trained policy is able to match or outperform the other methods across all the scenarios in terms of SLA satisfaction rate.

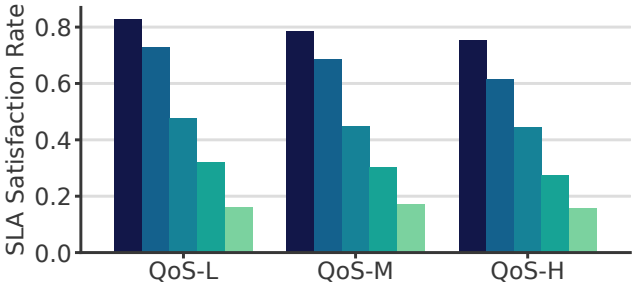
With respect to Herald, RELMAS achieves a geometric mean improvement of 59.4%, while with respect to PREMA-H it achieves a geometric mean improvement of 109%. The gratest improvements with respect to these two baselines are achieved for the Mixed workload. We believe, this effect should be attributed to high variability of model sizes in the Mixed workload, that cause the scheduling process to be more complex. In particular,



(A) Heavy Workload



(B) Light Workload



(C) Mixed Workload

FIGURE 5.4: SLA Satisfaction Rate comparison against other baselines for different workload sets.

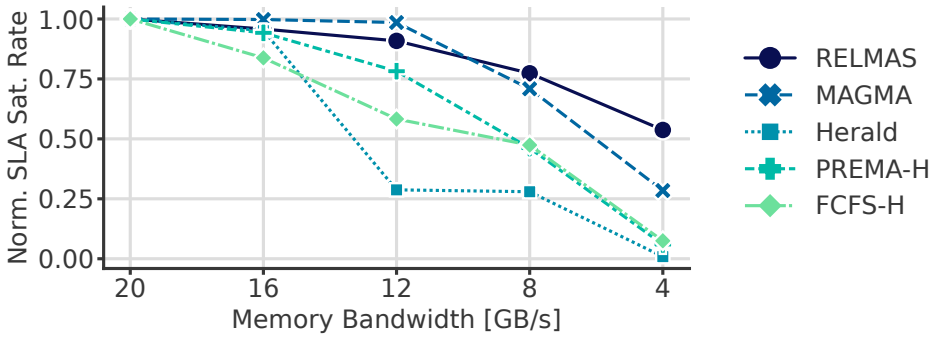


FIGURE 5.5: Impact of memory bandwidth reduction on SLA Satisfaction Rate for different scheduling strategies.

the Shortest-Job-First strategy of PREMA, allocates more priority to the Light models in the Mixed workload starving the Heavy models.

RELMAS achieves the same SLA Satisfaction Rate of MAGMA in the Light and Heavy workloads. However, it is worth noting that MAGMA is executed with same parameters of the original paper, i.e., 100 generations of 100 individuals, for each scheduling period to optimally scheduling a ready queue. For the evaluation of individual fitness, we employ the identical simulation platform, previously delineated and utilized for RELMAS training. These choices resulted in MAGMA scheduling runtimes of several minutes on a standard workstation for each simulated scheduling period, which is reasonable as MAGMA, unlike RELMAS, was born to be an offline scheduling policy, not an online one. While this places our MAGMA implementation in the realm of limited practicality within real-world contexts, it serves as a strong benchmark for evaluating the proposed approach. Furthermore, RELMAS is capable to outperform MAGMA in a Mixed workload workload scenario with up to 22.5%. SLA Satisfaction Rate improvement. We will analyze the overhead of the proposed solution in Sec. 5.5.3.

5.5.2 Bandwidth Sensitivity

RELMAS is designed to be aware of the bandwidth required by each DNN layer to be executed without stalls. To this end, this information is embedded as a feature in encoded LSTM policy input. This makes the proposed technique more robust in memory-constrained contexts. We test a Light workload scenario and in Fig. 5.5 we show the SLA Satisfaction Rate of different scheduling strategies normalized to their respective best result achieved. It can be noted that, independently from their absolute performance, some scheduling approaches tend to experience an abrupt degradation when decreasing the memory bandwidth, with respect to their performance in a bottleneck-free situation. On

the other hand, RELMAS degradation is slower, especially in the low-bandwidth region, indicating an higher capability of the proposed approach to better orchestrate layers on the heterogeneous hardware optimizing memory bandwidth utilization. We found that the same conclusions remain valid for the other considered workloads; therefore, we are reporting the results for the Light workload only.

5.5.3 Overhead Assessment

We now analyze the overhead of the proposed solution for the online scheduling in a multi-accelerator system. As already described, RELMAS consists in a DNN model composed of an LSTM cell followed by two projection fully connected layers. The policy works recurrently ingesting one input state encoding at each timestep, one for each layer in the ready queue. To gain insight into the computational complexity, it is informative to note that for a policy with an hidden size of 256, the number of Multiply-Accumulate (MAC) operations per layer amounts to 316 288, representing about a mere 0.04% and 0.007% of the MAC operations required by the AlexNet and ResNet50 models respectively. We opted to implement the scheduling policy within one of the accelerators designated for the policy's execution. This was achieved by communicating the accelerator's busy status to the scheduler, which was done by adding the overhead time to the first element in the State encoding, as shown in Fig. 5.2.5. Our experiments revealed that this implementation incurred no overhead impact on the scheduling process itself. We conducted an accurate evaluation of the energy overhead assuming that the scheduler is executed on the MAS itself. In particular, we assumed that one of the Simba Small SAs (Fig. 5.1) is responsible for running each timestep of the RELMAS policy. We evaluated the energy required for this setup using Timeloop [164]. We then assessed the overall energy overhead to the workload execution considering different hidden sizes for the LSTM cell and the Mixed workload. The results are shown in Fig. 5.6. We found that, for the considered workload, there is no significant SLA Satisfaction Rate improvement for an hidden size bigger than 128. In any case, the energy overhead does not exceed 1.3%. We also noticed that decreasing the scheduling period can increase the performance of the scheduler, because it enables shorter reaction times. This comes to the cost of a slight increase in energy overhead due to the increasing average length of residual ready queues. In other words, a layer will be scheduled multiple times on average before being actually executed, because it is often forwarded to the next scheduling period.

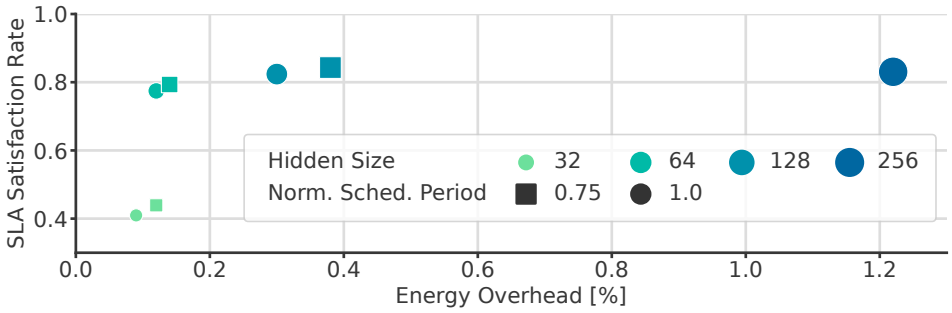


FIGURE 5.6: Energy overhead of the proposed scheduling algorithm varying the hidden size of the LSTM policy and the scheduling period.

5.6 Conclusion

This chapter introduced RELMAS, a low-overhead DRL-based online scheduling policy for DNN multi-tenant heterogeneous multi-accelerator systems. We believe that reinforcement learning could be a viable and low-overhead solution for online scheduling and dispatching in multi-accelerator multi-tenant heterogenous systems. For the proposed methodology, we included computation times and memory bandwidths requirements as features for the RL policy. We think many more could be explored to enable other functionalities. For instance, tenant-wise SLA satisfaction and fairness awareness could be enabled by exploring new state encoding and online tenant-wise deadline misses closed-loop control with dynamic reward shaping. Furthermore, we unconventionally considered an LSTM architecture for the policy but other trending architectures could be explored and their overhead assessed such as transformer-based models.

Chapter 6

Dataflow-Aware Online Scheduling for Graph Neural Network Inference

Graph Neural Networks (GNNs) have emerged as highly effective in fields such as recommender systems, bioinformatics, and network analysis. Nevertheless, the inherent irregularity of graph data poses distinct challenges for computational efficiency, prompting the design of specialized GNN hardware accelerators that outperform conventional CPUs and GPUs. Yet, the diverse structures of input graphs lead to variations in performance among different GNN accelerators, hinged on their specific dataflows. The impact of this performance variability, driven by diverse dataflows and graph characteristics, remains largely unexplored, which hinders the flexible deployment of GNN accelerators. In this chapter, we introduce GRAMAS, a data-driven framework tailored for latency prediction in GNN inference, taking into account dataflow sensitivity. Our methodology employs trained regressors to predict the latency of particular GNN inferences on varying dataflows, utilizing synthetic graph generation as a foundation. Experimental findings reveal that these regressors can predict the optimal dataflow for a given graph with an accuracy rate of up to 91.28% and a Mean Absolute Percentage Error (MAPE) of 3.78%. Furthermore, we present an online scheduling algorithm that leverages these regressors to optimize scheduling choices. Our experiments confirm that this algorithm achieves up to 3.17× speedup in mean completion time and 6.26× improvements in mean execution time relative to the best available baseline across all datasets. The research presented in this chapter was conducted in collaboration with N3Cat research group in Universitat Politècnica de Catalunya and was presented at the *31st Asia and South Pacific Design Automation Conference* (ASP-DAC 2025), held in Tokyo and published in the proceedings [175].

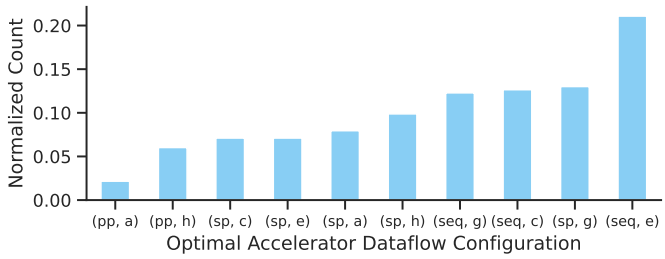


FIGURE 6.1: Latency-optimal GNN accelerator dataflow configuration (out of 24) distribution for ENZYMES [149] dataset.

6.1 Introduction

Graph data is inherently irregular when compared to other sources such as images or text. This irregularity introduces unique challenges related to the data movement and memory accesses involved in computing GNNs. These unique challenges have motivated the rise of specific GNN accelerator architectures which outperform conventional CPU and GPU acceleration, allowing for a much faster computation enabling low-latency GNN applications [1].

Despite the advancements in accelerator design for GNNs, the structural diversity of the input graphs still poses a substantial challenge. Variations in the graph's structure can lead to significant performance fluctuations across different dataflow strategies employed by accelerators. In Figure 6.1, we show the distribution of optimal dataflows in terms of latency for an example dataset among the considered possible configurations that will be introduced in the rest of this work. It can be seen that even for graphs belonging to the same domain, the selection of the best configuration is not trivial, and there is no one-size-fits-all solution.

In response to this issue, we propose a novel framework for data-driven dataflow-aware latency prediction for GNN inference. Our framework leverages lightweight parametric regressors trained on a large dataset of synthetic graphs to predict the latency of GNN inference under various dataflow strategies. This predictive capability allows for the optimization of GNN execution by selecting the most suitable dataflow strategy for a given graph structure. Additionally, we introduce an online scheduling algorithm that utilizes our latency prediction framework to dynamically allocate GNN inference tasks across heterogeneous accelerators.

We conduct comprehensive experimental evaluations to verify the effectiveness of our proposed solutions. On the one hand, we test the predictors across multiple axes by measuring the regression accuracy, ranking accuracy and the latency of a single configurable accelerator computing the GNN with the best dataflow choice according to the model. Experiments show that our model achieves up to 91.28% accuracy and 3.78%

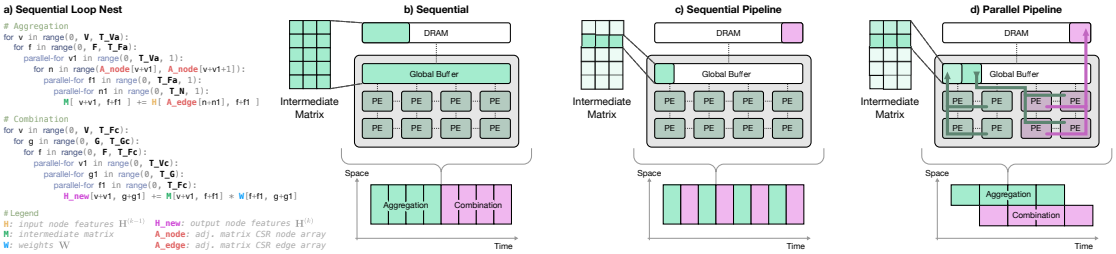


FIGURE 6.2: *Sequential*, *Sequential Pipeline* and *Parallel Pipeline* inter-phase dataflows and their respective memory usages. Loop nest representation of *Sequential* inter-phase dataflow.

regression MAPE in predicting the best configuration. This leads to a notable single accelerator improvement of up to 93.63%, a 58.42% improvement over the best fixed configuration for the dataset and 0.56% degradation over optimal on both real graph and synthetic datasets. On the other hand, we evaluate the performance of our online scheduling algorithm within a heterogeneous multi-accelerator environment where GNNs that arrive in a queue for computation are assigned to an accelerator. The experimental results reveal significant gains in scheduling performance with a 3.17× speedup in mean completion time, 6.26× speedup in mean execution time and more than 1000× speedup in turnaround time against the best feasible baseline on both synthetic and real-world datasets.

6.1.1 GNNs Inference Dataflows

Graph data is irregular when compared to image, audio and tabular sources. This motivated the rise of specific GNN accelerators [195, 233] featuring either fixed dataflows or adaptive capabilities, which outperform conventional GPU and CPU execution enabling high performance thanks to parallelization and energy efficiency due to high specialization.

Differently from traditional DNN workloads such as fully connected and convolutional layers [211], a layer of a GNN is organized in two phases and an *intra-phase dataflow* must be selected for each of them. Fig. 6.2a shows the loop nest representation of the main computation of a GNN layer. Aggregation and combination are implemented as two loop nests executed sequentially. The matrices involved are the same as in Fig. 2.7. In each phase, the intra-phase dataflow is specified by the loop ordering and the spatial loop tiling and unrolling. For the latter, the `parallel-for`s describe the parallel execution of a loop on several processing elements (PEs) by unrolling a specific dimension of the involved matrices. In the aggregation phase the tile sizes are `T_Va`, `T_Fa` and `T_N`, representing the tile sizes for the V, F and N dimensions respectively. The number of maximum utilized PEs is `T_Va`×`T_Fa`×`T_N`. However, notice that the N_v dimension is the

number of neighbours and depends on $v \in \mathcal{V}$, hence while iterating on low-degree nodes, the spatial accelerator may be underutilized with high `T_Ns`. In the combination phase, the loop nest represents a dense matrix multiplication and the tiling factors are `T_Vc`, `T_Gc` and `T_Fc`, representing the tile sizes for the V, G and F dimensions respectively. In this phase the static number of utilized PEs is equal to `T_Vc` $\times$ `T_Gc` $\times$ `T_Fc`

The global buffer and the PEs are interconnected through a Network-on-Chip (NoC) for data movement. According to the unrolling dimensions, data movement between PEs may be needed for spatial reduction, i.e., accumulation of partial sums, or multicast capabilities could be useful to reduce memory accesses. Generally, each dataflow choice requires specific microarchitectural implementations.

Furthermore, the dataflows of the two phases are interdependent. The interaction between the aggregation and combination phases is described by the *inter-phase dataflow* and determines the amount of memory accesses needed to move data from one phase to the other. In this work we adopt the sparse-dense workloads dataflows taxonomy proposed by Garg et al. [70]. Fig. 6.2 shows the space-time diagram of three different inter-phase dataflows:

- **Sequential**. With this configuration, the two phases are run sequentially similarly to two DNN layers. The output of the first phase (intermediate matrix) is written to memory and then loaded back to PEs for the next phase. The size of the intermediate matrix is $V \times F$, hence, as shown in Fig. 6.2, for large graphs the intermediate matrix cannot be stored entirely in the global buffer, requiring additional accesses to the off-chip DRAM causing higher energy consumption.

- **Sequential Pipeline**. Applying loop fusion and temporal tiling techniques to the Sequential loop nest, the execution of the Aggregation and Combination can be split in smaller steps temporally interleaved. As shown in Fig. 6.2, at each step a small tile of the intermediate matrix is calculated and stored in the global buffer or even in PE local buffers (if the two intra-phase dataflows are compatible [70]) to be used in the next phase step without accessing higher level memory.

- **Parellel Pipeline**. As shown in Fig. 6.2, the two phases can be also executed in parallel. This requires allocating a portion of the PEs to one phase and the rest to the other, the global buffer working as a ping-pong buffer and the NoC supporting the dual data movement. As in traditional general-purpose computing pipelines, the two stages are required to be balanced to avoid stall cycles. In particular the intermediate matrix tile production rate of the aggregation phase should be equal to the combination consumption rate. Depending on the specific graph input of the inference, this is not always possible, especially with heterogeneous node degree distributions.

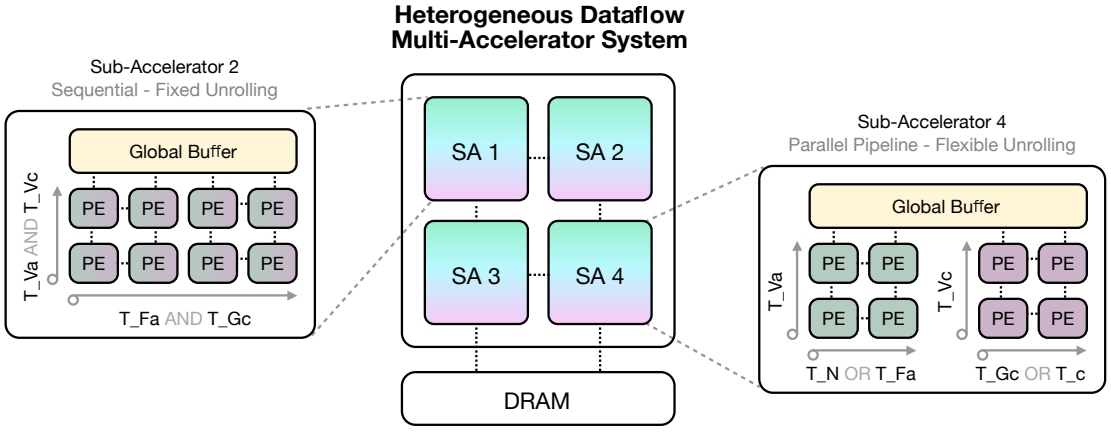


FIGURE 6.3: A multi-accelerator system for GNN inference.

6.1.2 Motivation

Despite advancements in accelerator design for GNNs, the main challenge in both flexible dataflow single-accelerator and multi-accelerator heterogeneous dataflow systems for GNNs remains the following: differently from DNN workloads, featuring fixed tensor sizes independent of the input, in GNNs, the amount of computation involved and consequently the size and structure of the matrices involved varies on the specific input graph (especially V and N_vs, i.e., number of neighbours for each node v). Variations in graph structures can cause substantial performance differences across dataflow strategies used by accelerators.

Thus, while for DNN workloads, an optimal dataflow choice for each layer can be determined offline and adopted in deployment for all the situational inputs [108, 144], for GNNs there is no one-fits-all solution. The optimal configuration of intra-phase dataflow and inter-phase dataflow has to be determined dynamically for each input dataset as it depends on the dataset characteristics. As shown in Figure 6.4, even for graphs of the same dataset, the optimal configuration in terms of latency varies, and different datasets exhibit different overall optimal configuration distributions.

The problem we aim to address in this work is predicting the latency of computing a GNN inference on a specific accelerator configuration. Having this knowledge prior to execution enables better computational decisions around GNN workloads. We are especially interested in using those predictions to rank configurations to select which is the best configuration for a given GNN.

For a more specific application, we consider the scenario of scheduling a set of jobs, namely GNN layers, each having a release time r_j (arrival time) not known in advance, in order to minimize the average completion times [171]. The objective is to address the real-time scheduling problem for heterogeneous multi-accelerator setups. This real-time scheduling scenario has been extensively investigated for traditional DNN workloads,

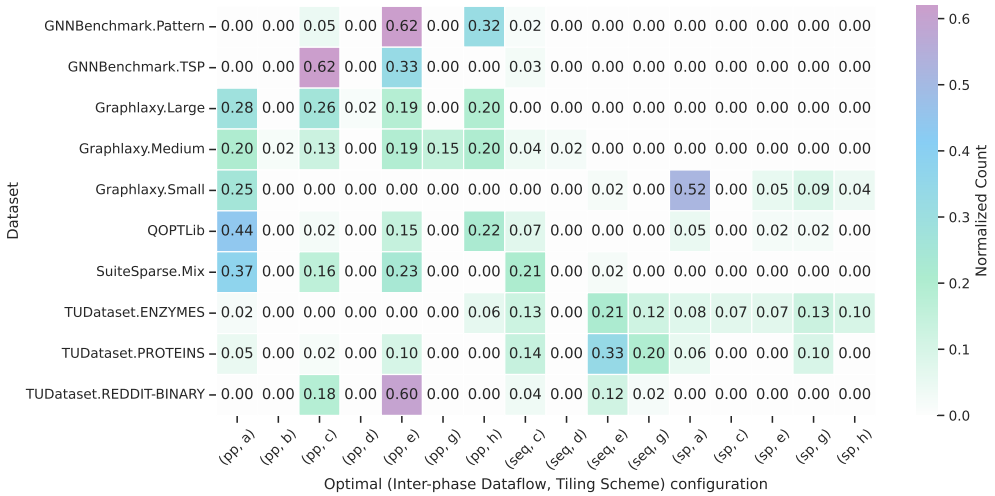


FIGURE 6.4: Frequency of optimal dataflow configurations.

with particular emphasis on Inference-as-a-Service and Multi-Tenant Data Center applications [22, 118]. The problem can be considered as unrelated parallel machine scheduling with job release times, which is NP-hard even in an offline setting and with arrival times known a priori [133, 198]. Leveraging the proposed latency predictor, machine-learning based speed-oblivious online scheduling techniques can be adopted [139].

6.2 Data-driven GNN Dataflow Selection

We address the challenge of predicting the latency of executing a GNN with a specific accelerator configuration (Section 6.1.2) by modeling dataflow selection as a learning problem. For an intra-dataflow $\omega \in \Omega$ and an inter-phase dataflow $\delta \in \Delta$ our objective is to select the set of parameters θ attaining the best approximation for the latency l of execution

$$\pi_\theta(\mathcal{G}, \omega, \delta) \simeq l(\mathcal{G}, \omega, \delta). \tag{6.1}$$

Knowing the latency before execution enables making dataflow-aware decisions for a collection of graphs. In this section, we first explore how to successfully find θ using supervised learning, and we then provide experimental results supporting our claims.

6.2.1 Learning to Predict Latency

In order to learn the GNN and configuration to latency mapping $\pi_\theta : \mathcal{G} \times \Omega \times \Delta \to \mathbb{R}$ effectively for any dataset, we require: (i) a significant amount of data with variability in the feature space of the graphs $\mathcal{G}$ topology, and (ii) the latency values for such GNN data across all accelerator configurations in $\Omega \times \Delta$. We satisfy the requirements by combining

TABLE 6.1: Proposed latency prediction model input features.

	Description		Symbol and/or Equation
base	Number of nodes and edges		V and E
	Spatial tiling factors		$\texttt{T\_Va}, \texttt{T\_Fa}, \texttt{T\_N}, \texttt{T\_Vc}, \texttt{T\_Gc}, \texttt{T\_Fc}$
	Graph density		$E \times V^{-2}$
	Clustering coefficient, measuring the probability that the adjacent nodes of a node are connected		[224]
	Seven node degrees quantiles normalized with respect to maximum degree including minimum degree		$Q_i, \quad \forall i \in \{1, \ldots, 7\}$
base+features	Number of operations, assuming a dense matrix mul. for the aggregation phase	S_1	$V \times F \times (G + \text{Mean Degree})$
	Estimation of the number of cycles for the combination phase	S_2	$\frac{V \times F \times G}{\texttt{T\_Vc} \times \texttt{T\_Fc} \times \texttt{T\_G}}$
	Estimation of the number of cycles for the aggregation phase (dense mat. mul.)	S_3	$\frac{V \times F \times \text{Mean Degree}}{\texttt{T\_N} \times \texttt{T\_Fa}}$
	Estimation of the latency for sequential inter-phase dataflow	S_4	$S_1 + S_3$
	Cycles estimation for aggregation phase assuming CSR encoding	S_5	$\sum_v^{\mathcal{V}} \frac{N_v \times F}{\min(N_v, \texttt{T\_N}) \times \texttt{T\_F}}$
	Cycles estimation for sequential inter-phase dataflow assuming CSR encoding	S_6	$S_3 + \frac{S_5}{\texttt{T\_Va}}$

two tools. For the graph level variability, we use Graphlaxy [224], a graph dataset generator where the resulting datasets are spread in the feature space of the graphs. We obtain the latencies of executing a graph across different accelerator configurations with STONNE-Omega [70], a framework for accurate simulation of the latency $\hat{l}(\mathcal{G}, \omega, \delta)$ of executing a graph $\mathcal{G}$ with an intra-dataflow ω and an inter-phase dataflow δ. By simulating the Graphlaxy graphs on STONNE-Omega we can generate a comprehensive dataset with variability in both $\mathcal{G}$ and ω, δ axis. Across the dataset, we select the parameters θ that satisfy

$$\arg\min_{\theta} [\pi_\theta(\mathcal{G}, \omega, \delta) - \hat{l}(\mathcal{G}, \omega, \delta)]^2. \tag{6.2}$$

Our model π_θ begins with a graph encoding step where metrics of interest are extracted from the graph $\mathcal{G}$. These metrics include conventional graph measures such as the number of nodes, density, and clustering coefficient, along with custom metrics engineered with the learning goal in mind (see Table 6.1 for a complete list of definitions).

TABLE 6.2: The 8 tiling schemes considered in this work.

Tiling	Aggregation			Combination		
ω	T_Va	T_Fa	T_N	T_Vc	T_Gc	T_Fc
a	$*$	$\min(\text{Num. PEs}, F)$	1	$*$	1	$\min(\text{Num. PEs}, F)$
b	$*$	$\min(2, F)$	$\lfloor F/2 \rfloor$	$*$	1	$\min(2, F)$
c	$*$	$\min(8, F)$	$\lfloor F/2 \rfloor$	$*$	1	$\min(8, F)$
d	$*$	1	1	$*$	1	1
e	$*$	$\min(18, F)$	$\lfloor F/2 \rfloor$	$*$	1	$\min(18, F)$
f	$*$	1	$\min(18, V)$	$*$	1	1
g	$*$	$\min(18, F)$	1	$*$	1	$\min(85, F)$
h	$*$	1	$\min(18, V)$	$*$	1	$\min(85, F)$

The encoding step is followed by a learnable non-linear transformation. We find gradient boosting methods to be the best-suited models as they provide accurate results with very fast training and inference. These methods proved particularly effective in handling the large variability of latency magnitude, outperforming other approaches, such as MultiLayer Perceptrons (MLPs), which required accurate data normalization to achieve comparable results. Additionally, we find that using separate models for each configuration results in more accurate predictions. By using this approach, we can effectively predict the latency of GNN executions across various accelerator configurations, allowing for optimized scheduling and resource allocation in heterogeneous computing environments.

6.2.2 Experiments on Latency Prediction

In the following paragraphs, we present a comprehensive evaluation of our approach to predicting GNN execution latency using datasets generated by Graphlaxy for training.

We experiment on 24 different configurations of STONNE-Omega given by the combination of the 3 inter-phase dataflows introduced in Sec. 6.1.1 and 8 tiling schemes. These schemes are reported in Table 6.2 and were selected based on insights from [70], considering different granularities and both spatial and temporal aggregation of neighbours. For all the tilings, the $*$ symbol indicates that PE utilization is maximized using T_Va and T_Vc. In our STONNE-Omega simulations, we employ the same accelerator architecture as in [70], comprising 512 PEs, each equipped with a 64B local register file.

We consider GCN [119] as the GNN model for our experiments. We use lightGBM [115] for the gradient boosting method implementation.

We propose two objectives for this area of experimentation. The first objective is to evaluate how well models trained on labels from executions of Graphlaxy graphs predict the latencies of unseen graphs coming from both Graphlaxy and other distributions, including graphs from real-world datasets. We evaluate the quality of the regressions

by measuring the MAPE. The lower the MAPE, the tighter the predictions obtained by the model. The second objective is to assess the impact of using the trained models to predict the best configuration for a single accelerator. In order to measure how good are the best configurations suggested by the model we measure the top-k accuracy of the predicted best configuration being on the top-k of true best accelerators. For deeper insight into what improvement would this selections achieve we compare the latencies of a flexible accelerator guided by our model against three benchmarks for alternative dataflow selection strategies: (i) random, selecting a random configuration for each graph; (ii) best fixed, selecting only one configuration which works best across the specific dataset; (iii) optimal, selecting the true best configuration for each graph.

We use three Graphlaxy datasets simulated on 24 STONNE-Omega configurations: small (243 graphs, avg. 10.49 nodes), medium (9,171 graphs, avg. 241.95 nodes), and large (6,381 graphs, avg. 782.36 nodes). We also include datasets derived from various real-world distributions, including 43 graphs representing real-world networks sourced from the SuiteSparse Matrix Collection [123], and 40 graphs from the QOPTLib [161] dataset, which encompasses quantum combinatorial optimization problem representations. Additionally, we consider 600 enzyme representations, 1113 protein representations, and a collection of 2000 graphs representing Reddit discussions from [149]. Furthermore, we include the Pattern and Travelling Salesman Problem (TSP) semi-synthetic datasets as described in [58]. The training and validation are done exclusively on the Graphlaxy graphs with the objective of obtaining models with low graph structure bias. We utilize a (70, 15, 15) split for training, validation and testing on the Graphlaxy datasets. All other non-Graphlaxy datasets are considered exclusively for testing.

Table 6.3 shows the experimental results in latency prediction metrics for the test sets. For regression quality, we observe a very small MAPE under 5% in medium and large in-distribution graphs, the error increases for out-of-distribution graphs and also for smaller graphs independently. The models are able to predict the best configuration out of the 24 possible configurations at different rates across different datasets. There are significant improvements from predicting the best configuration to predicting a Top-3 configuration, the magnitude of this improvements varies across datasets. We find that a smaller MAPE does not always lead to better Top-k accuracy; for instance, the TSP dataset has a larger MAPE than Enzimes but more than double the Top-k accuracy. The improvement of a model-guided flexible accelerator against random choices is large and consistent, over 80% across all datasets. Against the best fixed configuration, the model-guided flexible accelerator yields improvements for 8 out of 10 datasets. Naturally, fixed configurations show good performance for datasets with more homogeneous graphs, such as Enzymes and Pattern. Improvements over the best fixed configuration are also smaller for datasets with smaller graphs. Our method performs 5% or less from optimal in half of the datasets and under 15% from optimal in all datasets. The consistency in improvement over random and degradation over best is an indicator that, even when

Dataset Name	MAPE ↓ (%)	Top-1 acc. ↑ (%)	Top-3 acc. ↑ (%)	Improvement over random ↑ (%)	Improvement over best fixed ↑ (%)	Degradation vs optimal ↓ (%)
Graphlaxy.Medium	3.78	91.28	99.62	93.63	58.42	0.56
Graphlaxy.Large	4.83	83.63	98.20	96.27	8.99	0.86
Graphlaxy.Small	17.36	46.94	75.51	97.93	0.49	5.12
SparseSuite.Mix	13.32	62.79	81.40	89.79	8.04	5.24
QOPTLib	24.99	56.41	74.36	82.45	20.98	14.94
PATTERN	40.17	33.13	68.56	84.04	-6.30	9.40
TSP	29.13	62.18	92.85	87.38	2.18	4.22
ENZYMES	20.23	19.52	40.41	91.95	-3.29	12.10
PROTEINS	15.82	30.45	56.76	91.49	1.24	11.56
REDDIT-BINARY	14.16	35.08	76.45	85.96	0.74	8.68

TABLE 6.3: Performance of the proposed methodology in predicting GCN layer latency and identifying optimal dataflow configurations.

the models do not predict a Top-1 or Top-3 configuration all the time, there is a small performance difference between the predicted configuration and the best selection and a significant difference between the predicted configuration and a random choice.

Table 6.4 shows ablation results in justification of our modeling decisions. We observe that the base model requires some iteration in order to achieve the reported performance. Both regression and selection capabilities improve when adding synthetic features and when applying a logarithm transformation.

6.3 Online Scheduling

As introduced in the problem statement (Section 6.1.2), we present a specific real-world application for our methodology of learning to predict latency. Consider an online scheduling scenario where we want to schedule a set of jobs with unknown arrival times under a Pareto distribution into heterogeneous accelerators to minimize the average completion times [171]. In this section, we incrementally introduce an online scheduling strategy to leverage the latency prediction models from Section 6.2.1 along with a detailed experimental comparison against a selection of benchmarks.

6.3.1 Latency Prediction Guided Online Scheduling

In the context of online scheduling, the Shortest Job First (SJF) heuristic is particularly effective because it minimizes the average waiting time for jobs. By prioritizing shorter jobs, SJF ensures that more tasks are completed in a given time frame, thereby reducing the overall time jobs spend in the queue. This approach is beneficial in environments

TABLE 6.4: Ablation study on latency regression with added composite features (*+features*) and log-transformed target (*+log*).

Dataset Name	Model Name	MAPE ↓	Degradation over optimal (%) ↓
Graphlaxy.Medium	base	35.77	48.12
	base+features	10.76	4.06
	base+log	4.95	0.84
	base+features+log	3.78	0.56
SparseSuite.Mix	base	97.04	92.68
	base+features	31.68	12.52
	base+log	19.09	5.87
	base+features+log	13.32	5.24
QOPTLib	base	515.94	92.94
	base+features	152.67	86.60
	base+log	38.22	21.18
	base+features+log	24.99	14.94

with heterogeneous accelerators where job duration can vary significantly. The rationale behind SJF's effectiveness lies in its ability to keep the system's load balanced and prevent long jobs from delaying the completion of shorter ones. SJF can significantly outperform other scheduling algorithms in terms of minimizing average completion time, especially under high system loads and diverse job sizes [197].

In most real scenarios we do not know the length of the job before execution. In practice, we use estimates of the job length to construct approximate SJF variants. A simple approach in a graph context is to use the number of edges or nodes to determine the length of a job. Such information on the graph size is useful for load balancing between accelerators. Nevertheless, the previous approach has no information about the specific graph-accelerator performance. We can overcome this limitation by using the latency prediction models as estimators for the job length. This approach provides a more accurate dataflow-aware estimation of the job length.

6.3.2 Experiments on Online Scheduling

We assess the performance of online scheduling algorithms by measuring three metrics on jobs: (1) Mean Completion Time, when a job finishes; (2) Mean Turnaround Time, difference between finishing time and arrival time; (3) Mean Execution Time, difference between finishing time and starting time. We compare the following strategies to make scheduling decisions: (i) `Random`; (ii) `FCFS`, first come first served; (iii) `LIFO`, last come first served; (iv) `SJF-Nodes`, least nodes next; (v) `SJF-Edges`, least edges next; (vi) `SJF-Truth`, ground truth shortest job next; (vii) `Ours or SJF-Predicted`, the shortest job next based on latency predicted by the proposed model. Note that `SJF-Truth` is the

equivalent of the SJF heuristic introduced in Section 6.3.1 but is not feasible in practice since it requires knowledge about the true cost of executing a job in a processor. Our method serves as an approximation of SJF truth, which is feasible in practice.

We compose the online scheduling dataset for the experimentation. This dataset contains all the non-Graphlaxy graphs from the datasets from Section 6.2.2. The inter-arrival times are randomly drawn from a Pareto distribution, yielding a mean utilization rate of 85% across the included accelerator settings, and each graph corresponds to a job.

We consider a first scenario where we have a set of 3 accelerators (one for each inter-phase dataflow) and a flexible tiling configuration, similar to the one depicted in Figure 6.3. Thus, in this scenario, we need to decide which graph to schedule in which accelerator and also which tiling configuration that accelerator will act upon. We schedule the graphs from the online scheduling dataset to the accelerators using all the previous algorithmic strategies separately. The accelerator tiling decision for each graph is decided using the prediction for our `SJF-Predicted` method, the optimal tiling for the `SJF-Truth` and a random tiling for other baselines. Note that the knowledge of the optimal tiling makes the algorithm theoretical; it is not feasible in practice; we mark such algorithms with ✦.

Figure 6.5 contains the online scheduling results for this first scenario over five runs with a fixed seed for all algorithms. Metrics are normalized separately by the largest value in any algorithm. We observe that the combined effects of both having better scheduling decisions and a dataflow-aware tiling selection yield a large improvement over baselines in all metrics while matching the performance of `SJF-Truth` first with optimal tiling selection.

Next, we are interested in understanding what magnitude of the improvements is obtained by the tiling selections versus having better estimates for the graph cost. We consider the same heterogeneous multi-accelerator setting with the same algorithmic baselines with the difference that now all baseline algorithms select the optimal tiling after assigning a graph to an accelerator. For our method the tiling is still selected with the prediction.

Figure 6.6 shows the results for the second setting. With optimal tiling decisions the mean execution times and mean completion times of all baselines are very close to the true shortest job first performance. We observe that algorithms with better estimates for the graph cost significantly improve the turnaround time, indicating a superior ability to handle job queues and, consequently, better overall scheduling capabilities. Note that all methods using the true optimal tiling are not feasible in practice and thus marked with ✦.

Predictor Runtime Considerations The computational cost of inference for predictors is an important consideration for the practical feasibility of our approach. In

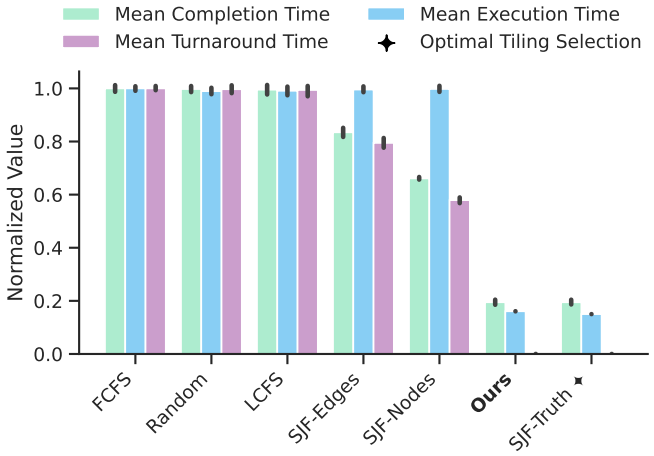


FIGURE 6.5: Algorithm performance for online scheduling policies with baselines using random tiling selection.

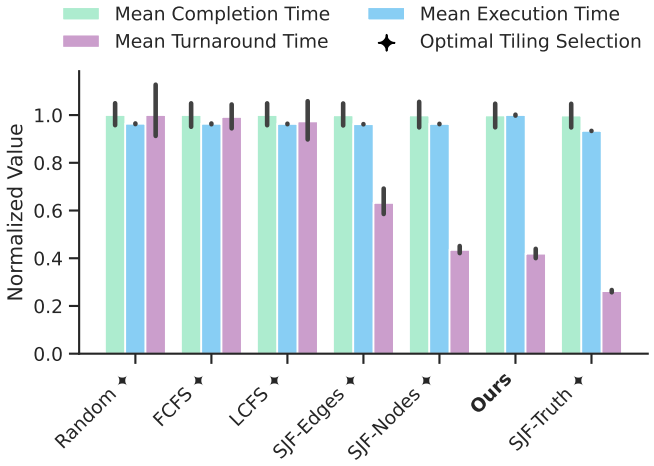


FIGURE 6.6: Algorithm performance for online scheduling policies with baselines using theoretical best tiling selection.

our implementation, we utilize a lightGBM boosting trees compiler called *lleaves* [23] to compile the trained models. To assess the inference time, we conducted measurements on a workstation equipped with a Ryzen 9 5900x processor. The average inference time for the latency predictor across all graphs in our datasets was 0.00232 microseconds. Assuming an accelerator operating at 1 GHz, and all the 24 models running in parallel on multiple cores, this translates to approximately $2,312$ accelerator cycles per inference. To contextualize this performance, we compare it to the average waiting time for job scheduling in our previous `SJF-Predicted` runs, which was $18,756.49$ cycles. The inference time for our predictive models is thus only about 12.3% of the average job waiting time. This comparison demonstrates the feasibility of implementing `SJF-Predicted` scheduling, even without dedicated GPU acceleration for the predictive models. It is worth noting that the computational cost of model inference remains constant regardless of graph size. Consequently, as graph sizes increase, the relative cost of model inference becomes increasingly negligible in the overall scheduling process.

6.4 Conclusion

GNNs present state of the art performance across domains with relational nature. Recent efforts have shown that different accelerators with GNN-specific dataflow architectures significantly outperform conventional GPU and CPU acceleration. In this work, we introduce a novel approach to tackling the unexplored question of how graphs with different properties perform on different dataflows. We train models to predict the latency of executing a certain graph on a certain dataflow on an extensive dataset of simulations. Our experimental evaluation shows that we can efficiently predict the best dataflow for a given graph. This expands the potential of adaptability of GNN accelerators to graph inputs. Moreover, we introduce an online scheduling algorithm leveraging the predictors. Experimental results show that our method outperforms all feasible baselines and matches the performance of the shortest job first, strong but unfeasible in practice, heuristic. Our method achieves up to 3.17× speedup in mean completion time, 6.26× speedup in mean execution time and more than 1000× speedup in turnaround time against the best feasible baseline on the online scheduling dataset.

Several intriguing research directions emerge for future work. The first is to expand on a larger design space exploration of the architecture. The second is to consider more complex models for latency prediction and online scheduling although this direction would require additional time cost concerns.

Part II

Quantum Computers

Chapter 7

Background on Quantum Computers

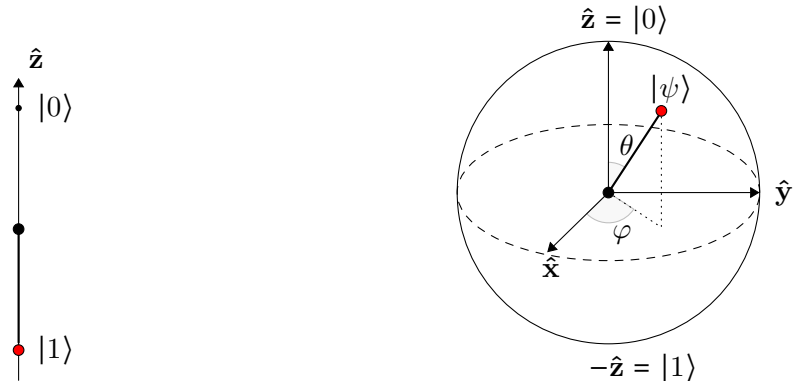
This chapter provides the conceptual and architectural background necessary for the remainder of the thesis. We begin by contrasting classical and quantum information, introducing single-qubit states and their geometric representation on the Bloch sphere. We then formalize multi-qubit states, tensor products, and the notion of entanglement, followed by a discussion of quantum gates, universal gate sets, and the quantum compilation (or transpilation) workflow—from high-level circuits to hardware-executable gate sequences with explicit qubit mappings. Finally, we describe architectural constraints, first in monolithic (single-core) devices with limited connectivity and then in emerging modular (multi-core) quantum systems where inter-core communication and state transfer become first-class optimization concerns.

7.1 Classical vs. Quantum Information

The basic information unit in classical information theory, i.e., the *classical bit*, can be in one of two discrete states: 0 or 1. This can be visualized as an "arrow" constrained to point either *north* (state 0) or *south* (state 1) along a single axis, as shown in Figure 7.1a. There are no intermediate states between 0 and 1, the bit must be one or the other. In contrast, a *quantum bit*, or *qubit*, can exist in any coherent superposition of the computational basis states $\{|0\rangle, |1\rangle\}$, expressed in *ket notation* [155]:

$$|\phi\rangle = \alpha\,|0\rangle + \beta\,|1\rangle = \alpha\begin{pmatrix}1\\0\end{pmatrix} + \beta\begin{pmatrix}0\\1\end{pmatrix} = \begin{pmatrix}\alpha\\\beta\end{pmatrix}, \qquad \alpha,\beta \in \mathbb{C}, \quad |\alpha|^2 + |\beta|^2 = 1. \tag{7.1}$$

The coefficients α and β are complex numbers constrained by normalization. This superposition is at the heart of quantum parallelism. For the so-called *observer effect* in quantum mechanics, upon measurement (or observation), the qubit collapses to $|0\rangle$ with probability $|\alpha|^2$ or to $|1\rangle$ with probability $|\beta|^2$. Before measurement, however, the



(A) Classical bit on a 1D axis.

(B) Qubit state as a point on the Bloch sphere.

FIGURE 7.1: Comparison between the information unit in classical and quantum computing.

qubit can evolve coherently, and quantum algorithms can exploit interference between amplitudes.

Any single-qubit pure state can (neglecting a global, physically irrelevant phase) be parameterized by two real angles, $\theta \in [0, \pi]$ and $\varphi \in [0, 2\pi)$:

$$|\psi\rangle = \cos\frac{\theta}{2}|0\rangle + e^{i\varphi}\sin\frac{\theta}{2}|1\rangle . \tag{7.2}$$

As illustrated in Figure 7.1b, this is equivalent to a point on the unit sphere, known as the Bloch sphere, with Cartesian coordinates

$$x = \sin\theta\cos\varphi, \quad y = \sin\theta\sin\varphi, \quad z = \cos\theta. \tag{7.3}$$

While a classical bit inhabits only the north or south pole ($|0\rangle$ or $|1\rangle$), a qubit can "point" in any direction on the sphere, representing a continuum of pure states. Single-qubit unitary operations correspond to rotations of this Bloch vector, mathematically, elements of SU(2), which project to SO(3) rotations up to an overall phase[1].

For n qubits, the joint state resides in a 2^n-dimensional complex Hilbert space. If $|\phi_1\rangle$ and $|\phi_2\rangle$ are single-qubit states, their joint (separable) state is the tensor product:

$$\begin{aligned} |\psi\rangle = |\phi_1\rangle \otimes |\phi_2\rangle &= \begin{pmatrix}\alpha_1\\ \beta_1\end{pmatrix} \otimes \begin{pmatrix}\alpha_2\\ \beta_2\end{pmatrix} \\ &= \alpha_1\alpha_2|00\rangle + \alpha_1\beta_2|01\rangle + \beta_1\alpha_2|10\rangle + \beta_1\beta_2|11\rangle . \end{aligned} \tag{7.4}$$

[1] SU(2) is the special unitary group of degree 2, consisting of 2×2 unitary matrices with determinant 1. SO(3) is the special orthogonal group in three dimensions, consisting of 3×3 orthogonal matrices with determinant 1.

In general,

$$|\psi\rangle = \sum_{b\in\{0,1\}^n} c_b |b\rangle , \qquad \sum_b |c_b|^2 = 1. \tag{7.5}$$

Not all multi-qubit states can be written as tensor products of single-qubit states. A state $|\psi\rangle$ is *entangled* if it is not separable, i.e., it cannot be expressed as

$$|\psi\rangle = |\phi_1\rangle \otimes |\phi_2\rangle \otimes \cdots \otimes |\phi_n\rangle .$$

A canonical example is the (maximally entangled) Bell state also known as Einstein, Podolski and Rosen (EPR) pairs:

$$|\Phi^+\rangle = \frac{1}{\sqrt{2}}(|00\rangle + |11\rangle). \tag{7.6}$$

No assignment of single-qubit states reproduces $|\Phi^+\rangle$. Entanglement yields non-classical correlations: measuring the first qubit of $|\Phi^+\rangle$ instantaneously determines the outcome distribution of the second (both collapse to the same classical value) while respecting causality. Entanglement is a key resource for quantum communication protocols as described in Section 7.5.1, superdense coding, and many quantum algorithms [155]. Conceptually, if we think of a qubit as a rigged coin, where measuring the qubit is like flipping the coin, then entanglement is analogous to rigging two coins together so that when both are flipped, their outcomes are correlated. In this way, the state of one coin is intrinsically linked to the state of the other, no matter how far apart they are.

Due to the fundamental principles of quantum mechanics, specifically the observer effect, any attempt to measure or observe a qubit inevitably disturbs its quantum state. This intrinsic property arises because the act of measurement causes the qubit to collapse from a superposition of states into one of its basis states. Unlike classical information, which can be duplicated or copied arbitrarily without any consequence, quantum information encoded in a qubit cannot be cloned or replicated exactly. This impossibility is formalized in the no-cloning theorem [228] and imposes fundamental constraints on quantum information processing.

7.2 Quantum Gates and Quantum Circuits

Quantum computation is performed by applying sequences of operations known as *quantum gates* to qubits that induce unitary evolutions of their quantum states[2]. A quantum program, also known as *quantum circuit*, consists of a set of target program qubits, namely *logical qubits*, and a sequence of gates. Each quantum gate corresponds to a *unitary matrix* U, acting on the state vector of one or more qubits. Unitarity ($U^\dagger U = I$)

[2]We focus on the gate-based quantum computational model [130]. Other computation models include quantum annealing [88] and digital-analog quantum computing [167].

ensures that quantum evolution is reversible and probability normalization is conserved. In general, the state of a multi-qubit system is represented as $|\psi\rangle \in \mathbb{C}^{2^n}$, where n is the number of qubits in the system. Consequently, a quantum gate can act on multiple qubits simultaneously and is represented by a $2^m \times 2^m$ unitary matrix, where $m \leq n$ is the number of qubits it acts on. An infinite amount of gates can be represented as unitary matrix operations, but only a subset can be practically implemented in quantum hardware. The subset implemented in a quantum computer is considered a *universal set* in the sense that every possible circuit can be reduced to a circuit featuring only gates of this subset, as a consequence of the Solovay–Kitaev theorem [120]. Usually, only 1-qubit and 2-qubit gates are implemented in hardware. Common single-qubit gates include: the Pauli operators,

$$X = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}, \qquad Y = \begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix}, \qquad Z = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix} \tag{7.7}$$

which correspond to 180° rotations about the x, y, and z axes of the Bloch sphere; the Hadamard gate H, which creates an equal superposition of $|0\rangle$ and $|1\rangle$:

$$H = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}, \qquad H|0\rangle = \frac{|0\rangle + |1\rangle}{\sqrt{2}} \tag{7.8}$$

and general rotation gates $R_x(\theta)$, $R_y(\theta)$, $R_z(\theta)$ rotating the state vector around x, y and z axis of the Bloch sphere respectively.

Entanglement can be generated using multi-qubit gates. The most fundamental is the *Controlled-NOT* (CNOT or CX), which flips the target qubit if the control qubit is in state $|1\rangle$. Its matrix representation is:

$$\mathrm{CNOT} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix}, \qquad \mathrm{CNOT}_{0\to1}|10\rangle = |11\rangle$$

Another particularly important two-qubit gate for actual circuit execution is the SWAP gate, which exchanges the states of two qubits:

$$\mathrm{SWAP} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}, \qquad \mathrm{SWAP}_{0,1}\left(\alpha|01\rangle + \beta|10\rangle\right) = \alpha|10\rangle + \beta|01\rangle$$

On many hardware platforms, SWAP is decomposed into three CNOTs:

$$\mathrm{SWAP}_{i,j} = \mathrm{CNOT}_{i\to j}\,\mathrm{CNOT}_{j\to i}\,\mathrm{CNOT}_{i\to j} \tag{7.9}$$

Quantum programs are typically represented using *circuit diagrams*, where time progresses from left to right and each horizontal line ("wire") represents a qubit. Gates are shown as boxes or symbols acting on the wires. For example, the diagram shown in Figure 7.2 depicts a circuit that prepares a Bell state by applying a Hadamard gate to the first qubit followed by a CNOT, creating the entangled state $|\Phi^+\rangle = (|00\rangle + |11\rangle)/\sqrt{2}$, then measurement is applied on the two qubits:

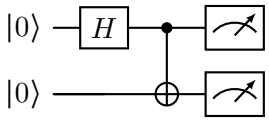


FIGURE 7.2: Bell state preparation circuit: Hadamard on the first qubit, then CNOT (control on the first, target on the second). Measuring both qubits yields perfectly correlated outcomes.

7.3 Physical Qubits

Analogous to the transistor's foundational role in realizing classical bits, diverse physical platforms have been engineered to embody qubits—the fundamental units of quantum information. The defining criterion for a physical qubit is its ability to function as a controllable two-level quantum system in accordance with the principles of quantum mechanics. Each technological approach brings distinct advantages and limitations, collectively shaping the landscape of contemporary quantum hardware.

Central challenges inherent to all physical qubit realizations include sustaining quantum coherence, suppressing operational errors during gate manipulations, and achieving high-fidelity state readout. Decoherence mechanisms, notably energy relaxation and dephasing, constrain the timescale over which quantum information can be preserved before irreversibly decaying to a classical state. Simultaneously, inaccuracies in control pulses and imperfections in measurement protocols can yield gate and readout errors. Additional complications such as environmental noise, qubit crosstalk, and architectural connectivity constraints intensify the difficulty of scaling quantum processors. To counteract these issues, various mitigation techniques are employed, including precise calibration of control schemes, dynamical decoupling protocols, and software-level error mitigation strategies [182].

A range of physical implementations has emerged, each with its own set of trade-offs. Among the most mature are superconducting circuits [122] (e.g., transmons and flux qubits), which currently underpin the quantum processors of industry leaders such as IBM [101], Google [79], Rigetti [183], and Alice&Bob [4]. These devices offer fast gate

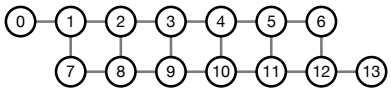


FIGURE 7.3: Example physical qubit coupling map.

operations and compatibility with scalable microfabrication techniques, but their coherence times are typically restricted to microseconds or milliseconds, and their coupling is generally limited to nearest neighbors in a planar lattice. Trapped-ion systems [151], in contrast, demonstrate exceptional coherence times and all-to-all connectivity within a chain, though they are challenged by slower gate operations and scaling complexities as the number of qubits increases. Neutral atom platforms [92], which exploit Rydberg interactions, enable flexible lattice geometries and programmable entanglement, yet current efforts are focused on improving gate fidelities and control precision. Spin qubits in semiconductors [26] offer potential for high-density integration and compatibility with established semiconductor processes, but maintaining coherence and device uniformity at scale remains a significant research endeavor. Topological qubits [121, 154] are theoretically endowed with intrinsic resilience to local noise, promising reduced error-correction overheads; however, achieving high-fidelity operations and scalable architectures is an ongoing challenge. Photonic qubits [174] are naturally suited for quantum networking and long-distance quantum state transmission, yet scalable on-chip integration for universal quantum computation continues to be a major hurdle.

In architectures such as superconducting and spin-qubit systems, the physical layout of qubits directly dictates which pairs can interact. The underlying hardware technology limits qubit couplings due to noise introduced by multiple interactions and spatial constraints of on-chip integration. The resulting *coupling graph* (also known as the *coupling map* or *topology*) specifies the set of physical qubits (nodes) and permitted interactions (edges) within a quantum processor. Figure 7.3 illustrates a representative coupling map for a 14-qubit IBM quantum device [101]. Efficiently mapping and routing logical qubits onto physical qubits is thus essential for practical quantum program execution on constrained hardware. Two-qubit gates, such as the CNOT, can only be implemented directly between connected qubits; otherwise, additional SWAP operations are necessary to reposition quantum information, in accordance with the no-cloning theorem. These auxiliary operations increase both circuit depth and the aggregate error rate, underscoring the importance of minimizing mapping overhead when compiling quantum circuits for real devices.

7.4 Quantum Compilation

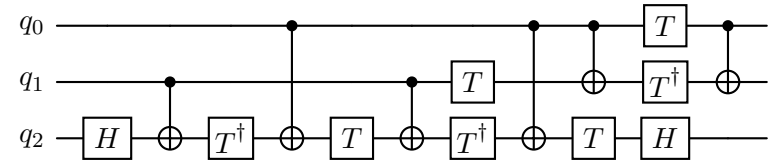
The *computing stack* describes the different layers of a computing system, from the physical implementation of computer chips, up through the control and instruction layers used to control and program the hardware, all the way to the application interface that enables users to interact with the system. In quantum computing, the computing stack is analogous to its classical counterpart. The quantum computing stack describes the different layers of a quantum system, from the elementary particles or devices used to store and compute information (the qubits), to the physical and logical systems that control and manipulate the qubits, all the way to the application layer, which includes high-level programming languages and algorithms used to program the quantum computer.

In the middle of this stack, quantum compilation, sometimes referred to as *transpilation*, is the process through which high-level quantum algorithms are transformed into hardware-executable instructions. This transformation is essential because most quantum algorithms are initially expressed in abstract terms, either as domain-specific programming languages or circuit descriptions, while quantum hardware can only execute a much more restricted set of operations, often with significant physical constraints as we outlined in Section 7.3. The compilation process bridges this gap by successively translating, optimizing, and adapting the input program to match the capabilities and limitations of the target quantum device.

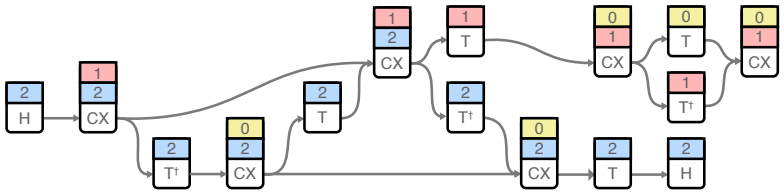
The journey from algorithm to hardware schedule typically begins with the decomposition of complex algorithmic primitives, such as the QFT or custom oracles, into a standard intermediate gate set. This stage enables subsequent, platform-agnostic optimizations. The next phase involves translating all gates into the native universal gate set supported by the hardware; for example, superconducting qubit platforms commonly use the CNOT as a multi-qubit gate combined with other single qubit gates. At this point, the logical structure of the algorithm must be *mapped* to the physical qubits available on the device. Careful qubit allocation is crucial because it influences the need for routing operations later in the process.

Due to the sparse and constrained connectivity (coupling graph) of most quantum hardware architectures, many two-qubit operations in the algorithm may involve qubits that are not directly coupled on the device as mentioned in Sec. 7.3. To enforce these architectural constraints, the compiler inserts movement operations such as SWAP gates, which bring the relevant qubits into proximity according to the hardware's coupling graph. These extra operations, while necessary, contribute to circuit depth and increase the overall error rate.

Once the algorithm is fully expressed in terms of native gates acting on specific hardware qubits, the compiler attempts to optimize the temporal arrangement of operations. By identifying gates that can be executed in parallel—subject to commutation relations and hardware limitations, circuit depth can be reduced, which is vital for mitigating



(A) Circuit diagram.



(B) Corresponding direct acyclic graph.

FIGURE 7.4: The quantum circuit representing a decomposed Toffoli gate.

the impact of decoherence. Advanced compilers also incorporate noise-aware strategies, such as routing logical qubits onto the most reliable physical qubits, preferring low-error CNOT orientations, and minimizing idle times for qubits. Additional techniques, such as template cancellation (removing redundant gate patterns), gate fusion (combining sequential gates into a single operation), and dynamic remapping, further enhance the compiled circuit's fidelity.

Ultimately, the quantum compilation process aims to produce an executable that minimizes a cost function that typically balances circuit depth (latency), the number of two-qubit gates (often the most error-prone operations), and the overall accumulated error probability, all while respecting the physical and architectural constraints of the quantum hardware. Through iterative translation, mapping, and optimization, quantum compilers play a crucial role in bringing abstract quantum algorithms closer to practical execution on noisy, intermediate-scale quantum devices.

In the following discussion, we concentrate on the mapping and routing stage of quantum compilation, commonly referred to as *layout synthesis* [138]. Figure 7.5 illustrates a representative compilation process for a decomposed Toffoli gate circuit mapped onto a 4-qubit grid coupling graph (see Fig. 7.4a). Initially, logical qubits are assigned to physical qubits in a manner that permits execution of the first CX_{q_1,q_2} gate as specified by the original circuit. However, this initial assignment does not enable execution of the subsequent CX_{q_0,q_2} gate, since q_0 and q_2 are mapped to p_2 and p_1, respectively, which are not adjacent and therefore cannot interact directly. To address this, SWAP gates are inserted to rearrange the mapping such that, whenever a two-qubit gate is required, the corresponding physical qubits are connected in the hardware topology. The introduction of these additional SWAP operations inevitably reduces overall fidelity, causing

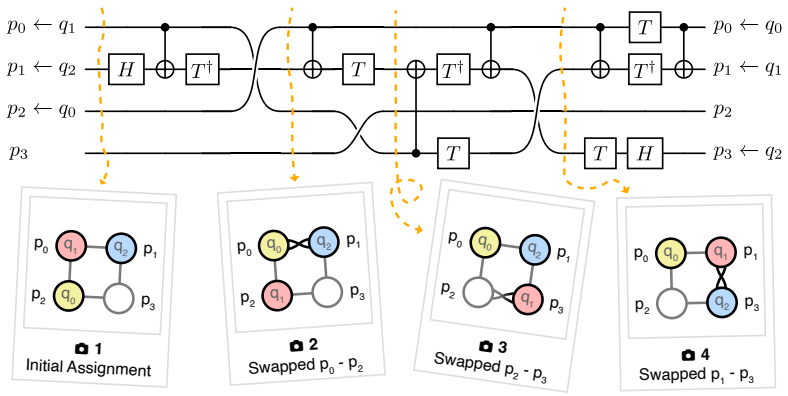


FIGURE 7.5: Compiled circuit diagram with additional SWAPs.

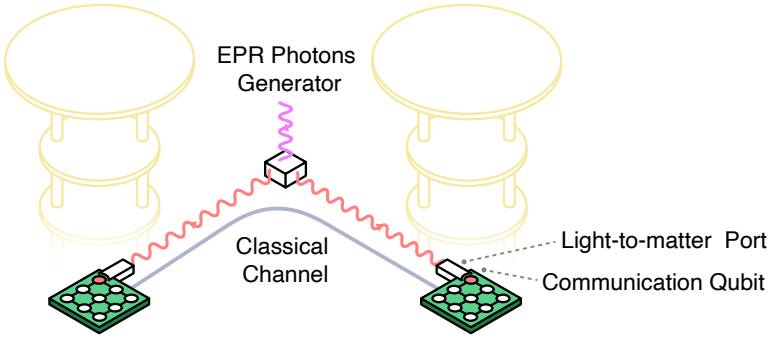


FIGURE 7.6: Multi-core quantum computer.

the executed circuit's output to deviate from the ideal theoretical result. Furthermore, the overall depth of the resulting circuit is increased, which directly correlates with the latency of circuit execution. This increased latency is particularly problematic given the limited coherence times of physical qubits, as it raises the probability that quantum information will be lost before computation is complete.

The qubit mapping problem of minizing the SWAP overhead is NP-complete as it relates to the well-known token swapping problem [24, 25, 203]. Many techniques to address the problem have been presented in literature, some of them will be mentioned in Section 7.6

7.5 Modular Quantum Systems

Scaling quantum computers presents unique challenges distinct from those encountered in classical computing. In classical systems, scaling typically involves miniaturizing components like transistors to pack more onto a chip, thus increasing performance and energy efficiency. By contrast, quantum engineering focuses on increasing the number of qubits, as the underlying physical processes are already operating at the quantum limit [130]. Crucially, a large qubit count is essential for quantum computers to solve meaningful, real-world problems [83, 147, 202], making scalability a fundamental requirement in quantum system design.

However, expanding current noisy intermediate scale quantum (NISQ) devices to accommodate more qubits introduces significant obstacles. For superconducting quantum computers in particular, these include maintaining qubit fidelity, operating at cryogenic temperatures to preserve coherence, managing the dense integration of control electronics, and mitigating crosstalk between qubits [9, 34, 196]. The conventional approach, integrating additional qubits into a single, monolithic array on one silicon substrate, exacerbates these issues by increasing wiring density, heightening crosstalk, and limiting manufacturing yield [204].

As depicted in Fig. 7.6, to address these challenges, researchers are increasingly looking to architectural strategies inspired by classical computing, such as dividing the quantum processor into multiple, smaller Quantum Processing Units (QPUs), also referred to as a multi-core approach [105]. In these systems, interconnects facilitate both classical and quantum communication for transferring quantum states between cores. As in classical multi-core or multi-chiplet architectures, inter-core communication in quantum systems is much more resource-intensive than intra-core operations and thus should be minimized. This is especially true for quantum systems, where transferring quantum states between cores can incur significant fidelity losses and introduce coherence-disruptive latency [184].

In multi-core quantum computing, the interconnect is also termed Quantum Network-on-Chip and connects multiple QPUs within a single system [27, 44, 63]. Within each core, qubit manipulation via local quantum gates is relatively efficient, while inter-core operations require specialized mechanisms due to quantum mechanical limitations such as the no-cloning theorem and the inherent difficulty of quantum state transfer. These interconnects may be implemented using direct qubit coupling, photonic links, or coherent quantum buses [48, 142].

This architectural paradigm offers several advantages to distributed systems. By reducing the need for long-range quantum communication, it provides a more practical short-term pathway toward scalable quantum computing compared to distributed approaches with multiple quantum computers interconnected via a quantum internet. The close proximity of QPUs allows faster, more reliable qubit interactions and reduces both decoherence and error rates [27]. Nevertheless, realizing the full potential of multi-core

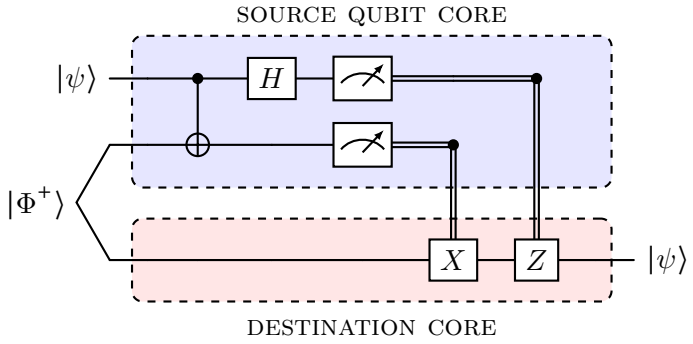


FIGURE 7.7: Qubit teleportation protocol realizing the teledata primitive. The two classical bits (from measuring the control/data at the source) select Pauli corrections at the destination.

quantum systems depends on overcoming significant technical barriers. High-fidelity inter-core communication must be achieved while minimizing noise and gate errors, and efficient strategies for qubit allocation and workload distribution are essential to maintain performance. Continued progress in quantum interconnect technologies and error correction will be vital for the scalability and feasibility of multi-core quantum architectures.

7.5.1 Inter-core Communication

In multi-core quantum architectures, it may be necessary to perform operations between logical qubits located in different cores. To facilitate such remote operations, appropriate communication primitives must be implemented [27]. Unlike classical distributed and modular systems, where information can be freely copied and transferred, quantum information is subject to the no-cloning theorem and must therefore be physically transported or teleported between cores. While direct qubit transfer using photons is theoretically feasible, it suffers from low success rates due to attenuation and decoherence. To address these limitations, quantum entanglement is leveraged to enable communication protocols based on quantum teleportation, which allow for the transmission of a qubit without requiring its physical movement.

In the following, we consider two fundamental communication primitives for inter-core interconnection, both of which rely on the quantum teleportation protocol. For a more comprehensive treatment of this topic, the interested reader is referred to [27]. Both communication primitives require a maximally entangled Bell pair distributed between the communicating cores, e.g., $|\Phi^+\rangle$. Distribution can be accomplished, for example, using a photonic EPR source that emits entangled photons to the two cores, where

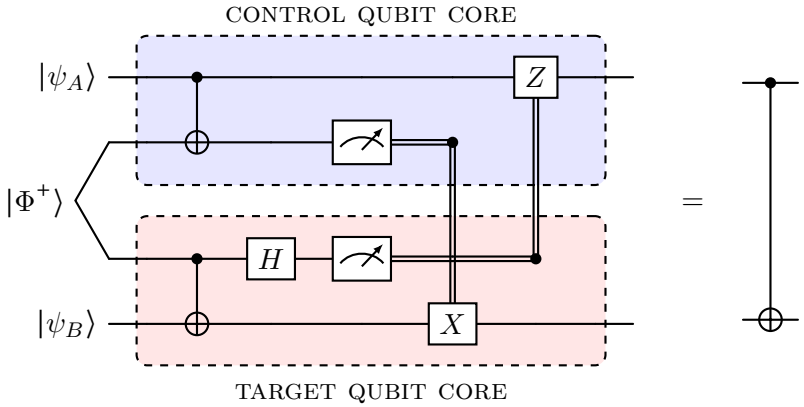


FIGURE 7.8: Teleportation-based telegate realizing a remote CX between data qubits residing on different cores.

light-to-matter interfaces map photonic states onto designated *communication qubits*; the remaining qubits are *data qubits*, as depicted in Fig. 7.9 [184].

Teledata The *teledata* primitive enables the transfer of a qubit from one core to another by leveraging a pre-shared Bell pair. To transmit the state of a data qubit, the qubit is first entangled with a local communication qubit at the source core. Both qubits are then measured, and the resulting outcomes are sent via classical communication to the destination core. At the destination, appropriate correction operations are applied to the remote communication qubit to faithfully reconstruct the original data qubit state. Specifically, if the measurement outcome of the communication qubit is one, a Pauli-X gate is applied to the communication qubit at the destination; subsequently, if the measurement outcome of the data qubit is one, a Pauli-Z gate is applied. A schematic depiction of this protocol is shown in Fig. 7.7. The classical communication step required to complete the protocol, often referred to as a *phone call*, ensures that quantum information transfer does not exceed the speed of light. Because the protocol relies on both the distribution of entangled pairs and the exchange of classical messages, the underlying hardware must support a core interconnection network capable of coordinating both quantum and classical information flow.

Telegate The *telegate* primitive facilitates the execution of a two-qubit gate between logical qubits residing on different cores, without necessitating the physical transfer of qubits [41]. Like *teledata*, this protocol relies on a pre-shared Bell pair between the participating cores. The procedure unfolds as follows: (i) two local CX gates are performed, each acting between a data qubit and its associated communication qubit at

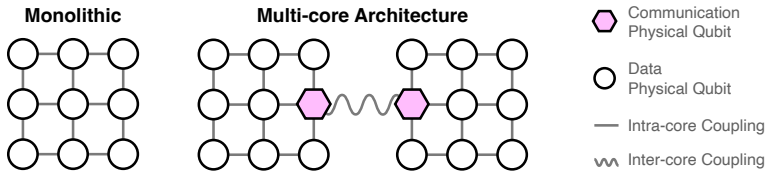


FIGURE 7.9: Monolithic and multi-core architecture physical qubits coupling graph comparison.

their respective cores; (ii) on the target core, a Hadamard gate is applied to the communication qubit; (iii) both communication qubits are measured, and classically controlled single-qubit corrections are subsequently applied to the remote data qubits based on the measurement outcomes. A circuit schematic illustrating a telegate emulating a direct CX operation between data qubits on separate cores is provided in Fig. 7.8. While the following discussion focuses on this specific telegate protocol, more advanced variants are possible. These include schemes that enable the execution of multiple gates using a single entangled pair, employing so-called *cat-entanglement* and *cat-disentanglement* circuits [27].

7.5.2 Compilation in modular architectures

Compared to the quantum compilation process for monolithic architectures discussed in Section 7.4, compiling circuits for modular, multi-core quantum architectures introduces additional layers of complexity. In such modular systems, the primary objective of the compilation process is to minimize the number of inter-core state transfers (*teledata*) and remote gate executions (*telegate*), as these operations incur significantly higher noise and latency overheads compared to local gates. Achieving this objective relies heavily on an effective qubit allocation strategy, wherein qubits are assigned to cores so as to maximize the number of gates that can be implemented locally within each core.

However, when the target quantum circuit requires interactions between qubits residing on different cores, inter-core communication becomes unavoidable. In these cases, one of the interacting qubits must be transferred to the core of its partner, or a remote gate must be executed using the appropriate communication protocol. Efficient scheduling and routing of these inter-core operations, especially in the context of complex multi-core topologies, present non-trivial challenges for the compiler.

Moreover, the inter-core communication primitives described previously are inherently multi-step and involve coordinated operations between data and communication qubits. If the intra-core qubit connectivity is not all-to-all, additional intra-core routing and layout synthesis may be required to enable the necessary interactions for inter-core protocols. As a result, the compilation workflow for modular architectures must address

not only inter-core communication minimization and scheduling, but also intra-core routing to support the underlying hardware constraints.

7.6 Related Works

Several works have addressed the quantum layout synthesis problem for single-core architectures with optimal and heuristic approaches [25, 98, 125, 135, 138, 152, 203] optimizing the number of intra-core swap operations and compiled circuit depth. In the following of the thesis, we focus on compilation techniques for multi-core quantum computers.

In addressing the multi-core qubit allocation challenge Baker et al. [13] introduced the FGP-OEE heuristic algorithm for paritioning qubits of a time-sliced circuit. A QUBO formulation of the problem was introduced in the work by Bandic et al. [15]. While envisioning a future resolution of this problem through quantum annealers, the current complexity lies in the tuning of the parameter tuning balancing feasibility and optimality in the objective, which can be a challenging and time-consuming task. Escofet et al. [61, 62] proposed a solution based on the Hungarian algorithm for assignments problems and derived theoretical bounds for the amount of inter-core state-transfers in all-to-all connected modular architectures. Cuomo et al. [43] introduced a formal problem formulation of the quantum compiling problem in distributed quantum systems focusing on remote gates, proposing a dynamic network flow model to minimize inter-core gates. Zhang et al. [239] introduced compilation for quantum chiplets considering inter-core communication through an *highway* of entangled qubits. Due to combinatorial nature of the problem, reinforcement learning is also being investigated as a potential way to derive heuristics [98, 173].

These methods focused on minimizing inter-core state transfers without considering the challenges of the teleportation protocol and the intra-core topologies or considered inter-core communication primitives different from teleport protocols. Leveraging teleport operations has also been proposed as a way to reduce swap overheads in single-core architectures [94, 162]. In [66] the authors propose a compiler for nearest neighbour distributed topologies taking into account entanglement swapping. In [67] the authors adopt a two-level optimization approach by first performing qubit assignement based on k-cut partitioning.

Chapter 8

Attention-Based Deep Reinforcement Learning for Qubit Allocation in Modular Quantum Architectures

In this chapter, we introduce a qubit mapping method for multi-core quantum computers that leverages reinforcement learning to minimize inter-core qubit transfers during algorithm execution. The research detailed in this chapter was presented at the *Design, Automation & Test in Europe* Conference (DATE) in Lyon, France, in April 2025 and is published in the proceedings [193].

In this work, we focus on reducing inter-core state transfer costs, assuming all-to-all connectivity within each core as shown in Fig. 8.1. Furthermore, we do not assume any specific inter-core state transfer protocol. As with previous works on qubit allocation optimization for both single-core [152] and multi-core [13, 15, 62] architectures, our approach begins by partitioning the input circuit into *slices*, where each slice contains only gates that can be executed in parallel—specifically, gates that do not share any logical qubits, as illustrated in Fig. 8.2a. As discussed in Section 7.5.2, for modular

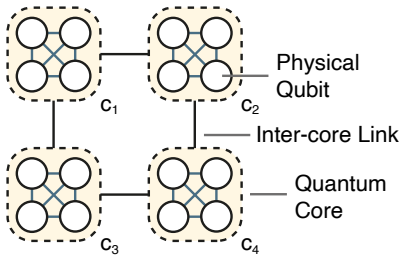
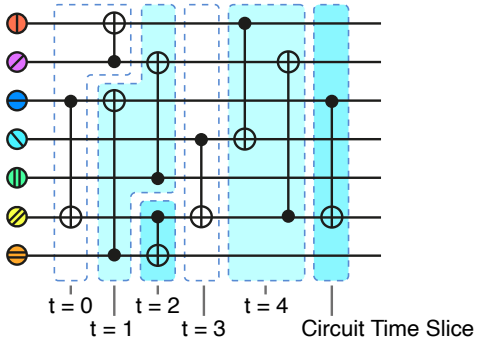
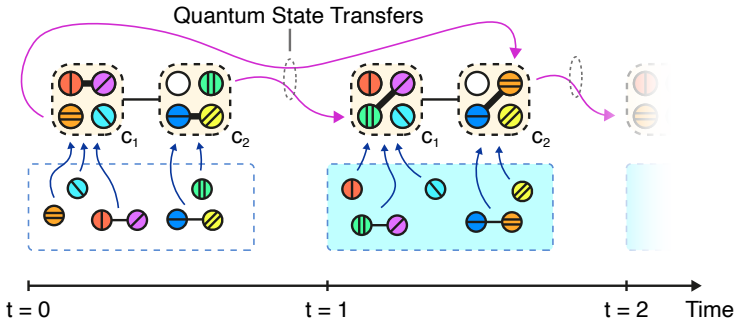


FIGURE 8.1: A multi-core quantum architecture with all-to-all intra-core qubit topology.



(A) A sliced quantum circuit.



(B) Example qubit allocation for first two slices.

FIGURE 8.2: Example execution scenario highlighting inter-core state transfers for qubit allocation for first two slices of a quantum circuit.

architectures, the subsequent step for each circuit slice involves allocating logical qubits to quantum cores while ensuring that: (a) *friend* qubits—those involved in the same gate—are assigned to the same core; and (b) the number of logical qubits assigned to each core does not exceed its physical qubit capacity. The goal of this allocation process is to minimize the number of inter-core communications required to transfer logical qubits from their locations in one time slice to their new locations in the following slice, as shown in Fig. 8.2b. In sparsely connected multi-core systems, the cost of transferring quantum states between cores can vary depending on the hop distance.

We present a novel approach based on an attention-driven Deep Reinforcement Learning (DRL) technique tailored to combinatorial optimization and sequential decision-making problems, which has demonstrated promise across various domains [37, 124]. To address the unique aspects of the multi-core qubit allocation problem, we utilize GNNs for state representation learning and design an action masking strategy to ensure

only feasible solutions are constructed. We validate the effectiveness of our attention-based agent by mapping random quantum circuits onto a 10-core system with a grid interconnection topology, benchmarking its performance against black-box optimization baselines. The main contributions of this study are as follows:

- We provide insights into the design of autoregressive DRL agents for the multi-core quantum compilation problem.
- We propose an attention-based agent capable of producing feasible solutions with deterministic execution time.
- We formulate and encode the problem for several derivative-free optimization algorithms, enabling a direct comparison between our learned heuristic and these baseline methods.

We now proceed by formulating the combinatorial optimization problem (Section 8.1), then we provide some background on autoregressive reinforcement learning applied to combinatorial optimization problems (Section 8.2). In Section 8.3 we describe the proposed solution in Section 8.4 we evaluate it against black-box optimization approaches and other methods proposed in literature. Section 8.5 concludes the chapter.

8.1 Problem Formulation

Formally, the problem addressed in this chapter is a combinatorial optimization problem that can be formulated as:

$$\min_{x} \quad \sum_{t=1}^{T-1} \sum_{q=1}^{Q} \sum_{s=1}^{C} \sum_{d=1}^{C} x_{t,q,s} x_{t+1,q,d} D_{s,d} \tag{8.1}$$

$$\text{s.t.} \quad x \in \{0,1\}$$

$$\sum_{c=1}^{C} x_{t,q,c} = 1 \qquad \forall q \quad \forall t \tag{8.2}$$

$$\sum_{q=1}^{Q} x_{t,q,c} \leq P_c \qquad \forall t \tag{8.3}$$

$$\sum_{c=1}^{C} x_{t,g_1,c} x_{t,g_2,c} \geq 1 \qquad \forall (g_1, g_2) \in G_t \quad \forall t \tag{8.4}$$

where x are the binary decision variables, $x_{t,q,c} = 1$ if at timestep t the q-th logical qubit is allocated in the c-th core, T is the number of time slices of the circuit, Q is the number of logical qubits, C is the number of cores, $D_{s,d}$ is the state transfer cost or distance between s-th and d-th cores, P_c is the capacity of the c-th core and G_t is the set of logical qubits pairs representing 2-qubit gates in the t-th slice, Eq. (8.2) constraints each logical qubit to be allocated in exactly one core in each time slice, Eq. (8.3) takes into account core capacities, Eq. (8.4) enforces in each slice and for each gate that qubits

involved are allocated in the same core, and the objective in Eq. (8.1) is the amount of inter-core communications. Similar formulation can be obtained for gates acting on more qubits. In this work we focus on 2-qubit gates as they are more frequently implemented in quantum architectures [16].

To obtain the partitioning of a quantum circuit in time slices the iterative Algorithm 2 can be considered. The algorithm operates on a given list G of 2-qubit gates, represented as pairs of logical qubit indices. For each gate (q_1, q_2), the process entails starting from the last time slice, moving backward until a slice t is encountered that contains gates involving either qubit q_1 or q_2. Subsequently, the gate is appended to slice $t+1$. If no such slice is identified, the gate is added to the first slice. If slice $t+1$ does not exist, it is created and appended to the output list of slices denoted as S.

Algorithm 2 Quantum circuit slicing procedure

Input : $G = [(g_1^1, g_2^1), \ldots, (g_1^L, g_2^L)]$; `#list of gates`
Output: $S = [G_1, \ldots, G_T]$; `#list of slices`
$T \leftarrow 0$
$S \leftarrow []$
foreach $(q_1, q_2) \in G$ **do**
  for $t \leftarrow T$ **to** -1 **do**
    if $t = -1$ **or** $q_1 \in \bigcup S[t]$ **or** $q_2 \in \bigcup S[t]$ **then**
      if $t + 1 > T$ **then**
        $T \leftarrow t + 1$
        S.append($\varnothing$)
      $S[t+1] \leftarrow S[t+1] \cup \{(q_1, q_2)\}$
      break

8.2 RL for Combinatorial Optimization

RL serves as a fundamental framework within the field of artificial intelligence, offering a paradigm that enables machines to learn and make decisions through interaction with an environment. At its essence, RL involves an agent, responsible for taking actions within a given context (state), and an environment, which responds by providing rewards or penalties based on these actions. The primary goal of the agent is to acquire an optimal policy, a strategic mapping of states to actions that maximizes cumulative rewards over time. This learning process, inspired by human-like trial-and-error mechanisms, has proven remarkably successful in diverse applications including robotics, control, games, autonomous driving and many others [74, 136, 208, 223].

Recent works have demonstrated promising results of RL applied to combinatorial optimization problems [17, 19]. As the solution space of combinatorial grows exponentially, it quickly become infeasible for a neural network to output a feasible solution in

one shot, due to the gargantuan action space. In our case, the number of possible decision variable assignments (including non-feasible solutions) for the problem described in Eq. (8.1) is equal to 2^{TQC}, e.g., about 10^{9030} when mapping a 30-slice circuit of 100 qubits on a 10-core architecture. To address these issues, similarly to language models [217], autoregressive methods that generates feasible solution step-by-step taking into account constraints have been recently proposed [19, 124]. In such methods the problem $\mathbf{x}$ is first encoded using a trainable encoder f_θ, obtaining an encoded representation $\mathbf{h}$:

$$\mathbf{h} = f_\theta(\mathbf{x}) \tag{8.5}$$

Then, the trained policy decoder g_θ outputs the best action probability distribution at each timestep t, based on the encoded representation of the problem and the current partial solution (result of the past actions). Formally we express this as follows:

$$a_t \sim g_\theta(a_t | a_{t-1}, \ldots, a_0, \mathbf{h}) \tag{8.6}$$

this is, at each timestep we select an action that keeps the solution valid, eventually obtaining a complete solution. Thus, given the problem $\mathbf{x}$, the policy π_θ outputs a probability distribution for a solution $\mathbf{a}$ built in T decoding steps:

$$\pi_\theta(\mathbf{a}|\mathbf{x}) \triangleq \prod_{t=1}^{T} g_\theta(a_t | a_{t-1}, \ldots, a_0, f_\theta(\mathbf{x})) \tag{8.7}$$

According to the RL paradigm, training the policy means finding the set of optimal parameters θ^* that maximize the expected return, namely the expected value of the reward given the distribution of policy actions and of the problem instances, formally [19]:

$$\theta^* = \arg\max_\theta \; \mathbb{E}_{\mathbf{x}\sim P(\mathbf{x})}\big[\mathbb{E}_{\mathbf{a}\sim\pi_\theta(\mathbf{a}|\mathbf{x})}[R(\mathbf{a},\mathbf{x})]\big] \tag{8.8}$$

In the following section, we describe how we applied this paradigm to the qubit allocation task in multi-core quantum architectures.

8.3 Methodology

In this section, we describe the proposed methodology for RL-based heuristic for the multi-core qubit allocation problem. Initially, we furnish an outline of encoding and decoding procedures in the trained policy. Subsequently, we introduce the components essential to represent and encode the input circuit and the environment state. Finally, we go deeper describing each component and we describe how we make sure that the output solution is a valid solution.

8.3.1 Autoregressive RL for Qubit Allocation

Autoregressive RL has been successfully applied to routing problems like the Travelling Salesman Problem (TSP) and Capacitated Vehicle Routing Problem (CVRP) [124], using an attention-based model. In TSP, the input is a set of coordinates, and the output is a permutation indicating the traversal order. Pointer Networks [221] help generate valid solutions by selecting the next city while masking visited nodes. Recently, these methods have been extended to domains with specific challenges, such as machine scheduling [38], user-server allocation [33], and hardware design [117]. Applying autoregressive RL to qubit allocation presents its own unique challenges.

Algorithm 3 Policy workflow

Input : $X = [G_1, \ldots, G_T]$; `#seq. of slices`
Output: $A = [A_1, \ldots, A_T]$; `#alloc. ∀ slice`
$A \leftarrow []$
$\mathbf{H}^{(I)} \leftarrow$ `InitEmbedding(`X`)`
$\mathbf{H}^{(S)} \leftarrow$ `EncoderBlocks(`$\mathbf{H}^{(I)}$`)`
$\mathbf{H}^{(X)} \leftarrow$ `AveragePooling(`$\mathbf{H}^{(S)}$`)`
for $t \leftarrow 1$ **to** T **do** `#for each circuit slice`
  $A_t \leftarrow []$
  $\mathbf{H}_t^{(C)} \leftarrow$ `SnapshotEncoder(`A_{t-1}`)`
  for $q \leftarrow 1$ **to** Q **do** `#for each logical qubit`
    context $\leftarrow$ `concat(`$\mathbf{H}^{(X)}$`,`$\mathbf{H}_t^{(S)}$`,`$\mathbf{E}_q^{(Q)}$`)`
    $\mathbf{f} \leftarrow$ current free capacities
    $\mathbf{d} \leftarrow$ distance from q's core in $t-1$
    $\mathbf{G}_{t,q}^{(C)} \leftarrow$ `DynamicEmbedding(`$\mathbf{H}^{(C)}$`, f, d)`
    $a \leftarrow$ `MaskedPointer(`context, $\mathbf{G}^{(C)}$`)`
    A_t.append(a)
  A.append(A_t)

The workflow of the trained policy, which takes an input quantum circuit and outputs a valid solution, is summarized in Algorithm 3. The policy receives circuit slices as input, and each slice t is encoded by `InitEmbedding` to generate embeddings $\mathbf{H}_t^{(I)} \in \mathbb{R}^{d_E}$, where d_E is the embedding size. Similar to [124], transformer `EncoderBlock`s capture relations between all slices in the circuit. The decoding process follows, where the policy sequentially assigns cores for each logical qubit in each slice, resulting in $T \cdot Q$ decoding steps (number of slices $\times$ logical qubits). The context for decisions includes three elements: the circuit representation $\mathbf{H}^{(X)}$, slice representation $\mathbf{H}_t^{(S)}$, and qubit representation $\mathbf{E}_q^{(Q)}$. The circuit representation remains constant, the slice representation stays the same for Q steps, and the qubit representation changes at each decoding step.

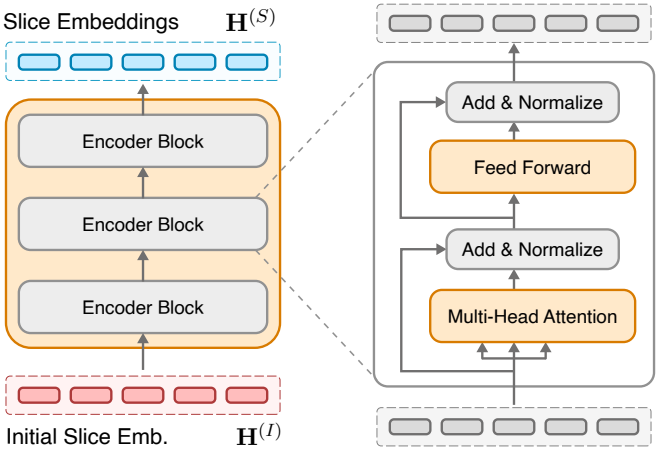


FIGURE 8.3: Attention-based circuit slice encoder.

The action space at each step is the set of possible cores for each qubit. Unlike single-encoder models in routing problems, we use a `SnapshotEncoder` to capture qubit allocations in previous slices. For each core c, an embedding $\mathbf{H}_c^{(C)}$ is generated, which includes information about the qubits allocated and core relationships. Additional dynamic data, such as core capacity and transfer costs from previous slices, are incorporated at each step. A masked attention-based pointer mechanism selects the core for each qubit, ensuring valid allocations and feasible solutions for the next steps. After decoding, the solution assigns cores for all qubits in each slice while satisfying problem constraints.

8.3.2 Circuit Slice Encoder

The first step of the autoregressive policy is obtaining a learned representation of the input circuit, as shown in Eq. (8.5). In traditional RL problems modeled as a Markov Decision Process (MDP), the agent makes decisions based on the current state, derived from past states and actions. In autoregressive combinatorial optimization, however, a global representation of the input is built first, followed by sequential decisions focusing on specific contexts. Our encoder constructs a slice representation incorporating global circuit information. Fig. 8.3 shows the circuit slice encoder, using attention-based encoder blocks from [217].

8.3.2.1 `InitEmbedding`

In this regard, the challenge specific to the qubit allocation problem is that we first need to represent the input of the transformer encoder as a set of initial embeddings

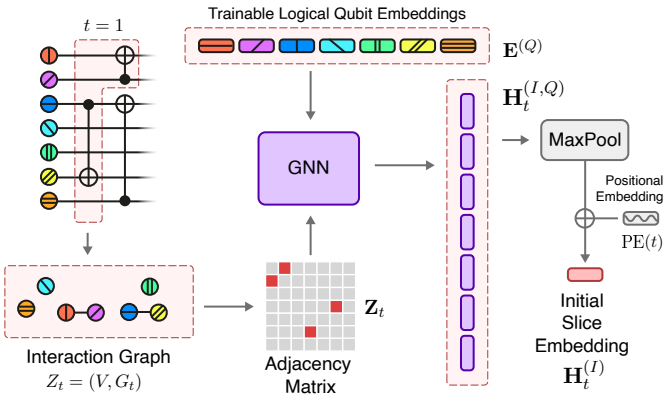


FIGURE 8.4: Initial embedding of the time slice $t = 1$ through a GNN layer.

$\mathbf{H}_t^{(I)} \in \mathbb{R}^{d_E}$ for each slice t where d_E is the size of such embedding. The output of the slicing Algorithm 2, described previously, is the sequence of the T slices we need to embed. Each slice G_t consists of a set of logical qubit pairs representing the gates contained in the slice. We can consider G_t as the edge set of a undirected disconnected graph $Z_t = (V, G_t)$ representing the interaction between logical qubits V given by gates [15]. Because of this, we decide to embed each slice employing a GNN, as shown in Fig. 8.4. In particular we employ the convolutional graph operator introduced in [119]. In our case, for each slice t, we obtain a qubit-wise representation $\mathbf{H}_t^{(I,Q)} \in \mathbb{R}^{Q \times d_E}$ as follows:

$$\mathbf{H}_t^{(I,Q)} = \tilde{\mathbf{D}}_t^{-\frac{1}{2}} \tilde{\mathbf{Z}}_t \tilde{\mathbf{D}}_t^{-\frac{1}{2}} \mathbf{E}^{(Q)} \mathbf{W}^{(I)}, \tag{8.9}$$

where $\mathbf{W}^{(I)}$ is a trainable weight matrix, $\mathbf{E}^{(Q)} \in \mathbb{R}^{Q \times d_E}$ are trainable logical qubit embeddings, $\tilde{\mathbf{Z}}_t \in \{0,1\}^{Q \times Q}$ is the adjacency matrix of the interaction graph Z_t with added self-loops and $\tilde{\mathbf{D}}_t^{-\frac{1}{2}}$ the element-wise reciprocal square root of its diagonal degree matrix. On the obtained qubit-wise embeddings $\mathbf{H}_t^{(I,Q)}$ we apply a max pooling across the qubit dimension Q. Then, sinusoidal positional encoding [217] is applied finally obtaining the embedding $\mathbf{H}_t^{(I)} \in \mathbb{R}^{d_E}$ for each circuit time slice t.

8.3.2.2 `EncoderBlocks`

The initial slice embeddings $\mathbf{H}^{(I)} \in \mathbb{R}^{T \times d_E}$ are processed by b transformer encoder blocks [33, 124, 217], which incorporate information from other circuit slices. This allows the policy to consider future slices when selecting cores for qubits, improving decision-making. The transformer model outperforms conventional neural transduction models like recurrent and convolutional networks by capturing long-range dependencies,

making it ideal for large-scale problems and parallel execution. Each encoder block includes a Multi-Head Attention (MHA) layer and a fully connected (FC) layer, both followed by residual connections and batch normalization. The attention mechanism, which computes outputs as weighted sums of input value vectors based on the compatibility of query and key vectors, will be important in the decoding phase. For more on the encoder block, see [217]. The scaled-dot-product attention function is defined as follows:

$$\text{Attention}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{softmax}\left(\frac{\mathbf{Q}\mathbf{K}^\top}{\sqrt{d_K}}\right)\mathbf{V} \tag{8.10}$$

where $\mathbf{Q}$ are the query vectors, $\mathbf{K}$ are the key vectors and $\mathbf{V}$ value vectors. d_K is the dimensionality of the key vectors. $\mathbf{Q}$, $\mathbf{K}$ and $\mathbf{V}$ are obtained through 3 different fully connected layers (linear projections) applied on some input. If the queries $\mathbf{Q}$ and the key-value pairs $\mathbf{K}$, $\mathbf{V}$ are calculated starting (projecting) from the same input vectors, the attention is named self-attention. If $\mathbf{Q}$ is calculated on a different input, it is called cross-attention. The MHA calculation consists in combining the result of the attention from different attention *heads* as follows:

$$\mathbf{H}_i = \text{Attention}(\mathbf{Q}\mathbf{W}_i^Q, \mathbf{K}\mathbf{W}_i^K, \mathbf{V}\mathbf{W}_i^V) \tag{8.11}$$

$$\text{MHA}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{concat}(\mathbf{H}_1, \dots, \mathbf{H}_h)\mathbf{W}^O \tag{8.12}$$

where $\mathbf{W}$ are linear projection trainable weights and h is the number of heads. In the first encoder the MHA is applied in self-attention mode to the projections of the initial slice embeddings, incorporating in each slice embedding information from other *compatible* slices. Subsequent encoder blocks perform the same operation on the output of the previous encoder block. The output of the b-th (last) encoder block represents the ultimate slice embeddings $\mathbf{H}^{(S)} \in \mathbb{R}^{T \times d_H}$, where T is the number of circuit slices and d_H is the dimensionality of the embeddings, considered in the decoding phase.

8.3.3 Core Snapshot Encoder

The task of the core `SnapshotEncoder` is to provide an encoding of the qubit allocations in the previous circuit time slice. During the decoding process, in fact, we also want the policy to take into account the previous qubit allocation, possibly trying to make a compromise between the state transfer distances from the previous slices and the qubit allocation for the next slices still to be decided.

During the decoding, when processing the t-th time slice, the output action for the previous slice $t-1$ is a vector $A_{t-1} \in \{1, \dots, C\}^Q$, i.e., it consists in the mapping between each logical qubit and the core index. As shown in Fig. 8.5, the Core Snapshot Encoder transform the allocation it represents in the embeddings $\mathbf{H}_t^{(C)} \in \mathbb{R}^{C \times d_H}$. This

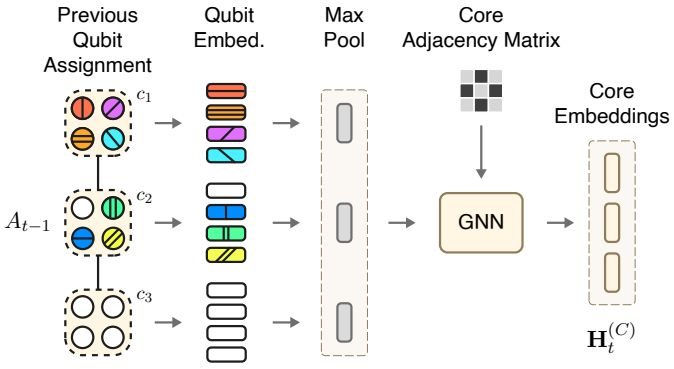


FIGURE 8.5: Previous slice qubit assignment snapshot encoder.

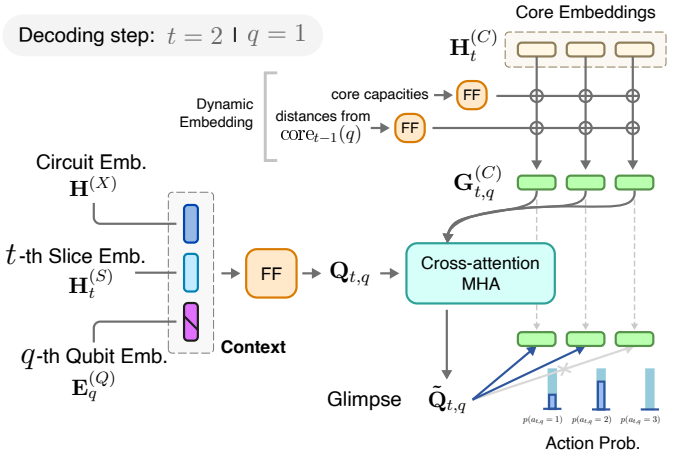


FIGURE 8.6: Action probability calculation at each decoding step.

is obtained employing another GNN similarly to the `InitEmbedding` component. This time the input adjacency matrix is fixed and represents the connectivity of the multi-core hardware architecture. Regarding the node input features, instead, these are obtained for each core by performing the max pooling of the embeddings of the logical qubits allocated on it. Logical qubit embeddings are calculated for each physical qubit available in the core. If a physical qubit is not associated to logical qubit, a padding embedding is considered.

8.3.4 Decoding Process

As described earlier and summarized in Algorithm 3, the decoding process spans $T \times Q$ steps, where each logical qubit in each slice is allocated sequentially. Each decoding step

is identified by the indices (t, q), representing the current slice and logical qubit, respectively. The context at each step is the concatenation of three embeddings: (a) the circuit embedding $\mathbf{H}^{(X)}$, calculated as the average of slice embeddings from the circuit slice encoder; (b) the slice embedding $\mathbf{H}_t^{(S)}$ for the current slice; (c) the trainable embedding $\mathbf{E}_q^{(Q)}$ of the current logical qubit.

At the beginning of each new slice t, the qubit allocation snapshot from the previous slice $t-1$ is encoded into embeddings $\mathbf{H}_t^{(C)} \in \mathbb{R}^{C \times d_H}$ by the `SnapshotEncoder`. For each logical qubit q, these embeddings are augmented via the `DynamicEmbedding` process. The remaining core capacities are projected into embeddings of dimensionality d_H and summed with $\mathbf{H}_t^{(C)}$. Core capacities are reset at the start of each slice and decrease as qubits are allocated. Similarly, the distance of qubit q from each core, based on its previous allocation and the architecture's core distance matrix, is also projected into embeddings and summed with the core embeddings. Finally, from this `DynamicEmbedding` procedure, we obtain core embeddings $\mathbf{G}_{t,q}^{(C)} \in \mathbb{R}^{C \times d_H}$ that incorporate information about: (a) qubit allocation in the previous slice; (b) current remaining capacity; (c) hypothetical state transfer cost for qubit being mapped;

As shown in Fig. 8.6, the final step of each decoding step is obtaining the probabilities of allocating the qubit q of slice t on each possible core c, starting from the current context and the core embeddings $\mathbf{G}_{t,q}^{(C)}$. This task is achieved with an attention mechanism [124] and it first involves projecting the context to obtain a query vector as follows:

$$\mathbf{Q}_{t,q} = \text{concat}\left(\mathbf{H}^{(X)}, \mathbf{H}_t^{(S)}, \mathbf{E}_q^{(Q)}\right)\mathbf{W}^{(V,G)} \tag{8.13}$$

where $\mathbf{W}^{(V,G)}$ is the trainable weight matrix for query projection from the context. Then, the cross attention is performed between the query and the core embeddings obtaining the attended query vector $\tilde{\mathbf{Q}}_{t,q}$ also known as *glimpse* [17, 220]:

$$\mathbf{K}_{t,q} = \mathbf{G}_{t,q}^{(C)}\mathbf{W}^{(K,G)} \tag{8.14}$$

$$\mathbf{V}_{t,q} = \mathbf{G}_{t,q}^{(C)}\mathbf{W}^{(V,G)} \tag{8.15}$$

$$\tilde{\mathbf{Q}}_{t,q} = \text{MHA}\left(\mathbf{Q}_{t,q}, \mathbf{K}_{t,q}, \mathbf{V}_{t,q}\right) \tag{8.16}$$

where $\mathbf{W}^{(K,G)}$ and $\mathbf{W}^{(V,G)}$ are the trainable weights matrix projecting key and value vectors from the augmented core embeddings. Subsequently, compatibilities are calculated between the attended query and the core embeddings, similarly to the attention function in Eq. (8.11). Compatibilities with cores on which mapping the current logical qubit q would result in an invalid solution are masked, thus having:

$$u_{t,q,c} = \begin{cases} -\infty, & \text{if masked} \\ \frac{\tilde{\mathbf{Q}}_{t,q}\mathbf{K}_{t,q}^{\top}}{\sqrt{d_K}}, & \text{otherwise} \end{cases} \tag{8.17}$$

The resulting vector $\mathbf{U}_{t,q}$ can be interpreted as logarithm of probabilities (logits) and the final output action probability for the (t, q) decoding timestep can be computed using the softmax function:

$$p(a_{t,q} = c) = \frac{e^{u_{t,q,c}}}{\sum_{j=1}^{C} e^{u_{t,q,j}}} \qquad \text{for } c = 1, 2, \ldots, C \tag{8.18}$$

Next section describes when a core (action) is masked and cannot be selected.

8.3.5 Action Masking

Appropriately masking logits in Eq. (8.17) is crucial for building a valid solution, as it depends on the current partial solution from previous actions. The first masking mechanism addresses the core capacity constraint in Eq. (8.3). A core's logit is masked if its remaining capacity is 0, preventing further qubit allocation in the current time slice. Another necessary masking applies to the *friendship* constraint in Eq. (8.4), which involves gate interactions between logical qubits. During decoding, qubits are allocated sequentially. If two qubits q_1 and q_2 are involved in the same gate, q_1 is allocated first, and when decoding q_2, only the core of q_1 is allowed. If core capacity is exhausted before allocating q_2, the core's capacity is reduced by 2 when allocating q_1, reserving space for q_2, but no capacity is reduced when q_2 is actually allocated.

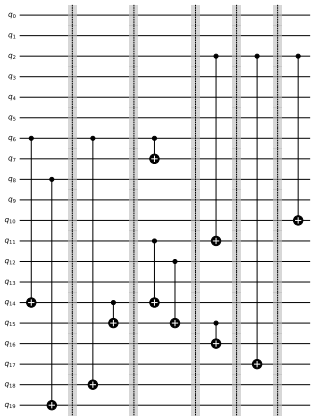
One more issue arises in a scenario with 2 cores, each having 2 physical qubits, and a circuit with four qubits q_1, q_2, q_3, and q_4, where $q_3 \bullet\!\!-\!\!\bullet\, q_4$ are involved in the same gate while q_1 and q_2 are non-interacting. If q_1 and q_2 are allocated to different cores before $q_3 \bullet\!\!-\!\!\bullet\, q_4$, both cores may have insufficient capacity to allocate the gate. To prevent this, when allocating a qubit, the number of remaining interacting qubit pairs g is considered, and only cores with sufficient capacity to allocate pairs (i.e., $\sum_{c=1}^{C} \lfloor \text{capacity}_c/2 \rfloor \geq g$) are not masked.

8.3.6 Reward and Training

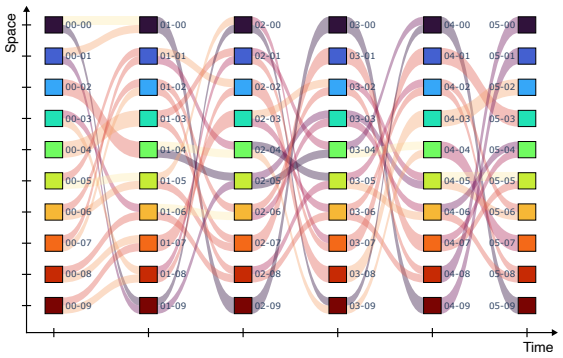
According to the problem formulation in Sec. 8.1, after building an overall solution sampling the probability distributions at each decoding step, the corresponding reward is calculated as the total number of inter-core communications needed to bring a qubit in its assigned core from the core where it was allocated in the previous circuit slice. Formally, the reward is defined as:

$$R(A) = \sum_{t=2}^{T} \sum_{q=1}^{Q} D_{A_{t-1,q}, A_{t,q}} \tag{8.19}$$

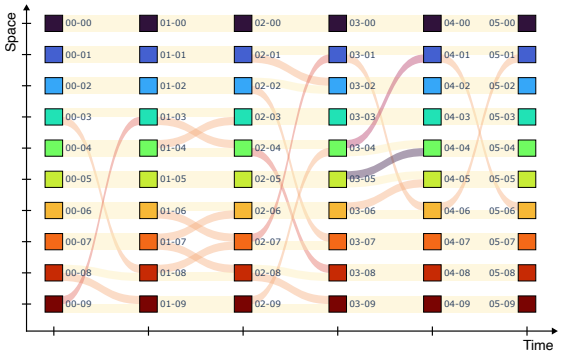
where D is the distance matrix and $A_{t,q}$ is the core (action) selected for q-th qubit at t-th slice. The reward is formulated in this to equal the total amount of state transfers



(A) Sliced quantum circuit.



(B) Untrained policy qubit allocation.



(C) Trained policy qubit allocation.

FIGURE 8.7: Inter-core state transfers visualization of two allocation strategies for a 20-qubit 6-slice quantum circuit on a 10-core × 2-qubit system featuring a grid topology. In the two Sankey diagrams, each row represent a core, each column is a time slice. A link represents a quantum state transfer. Darker links signify state transfer between distant cores in the grid.

needed for a given allocation solution. The training loss is then defined as the expected value of $R(A)$ and optimized by gradient descent using the REINFORCE [226] gradient estimator with rollout baseline as in [124].

8.4 Evaluation

In this section we provide implementation details and evaluation results of the proposed methodology.

8.4.1 Experimental Setup

We implemented the proposed method using PyTorch [168], PyTorch Geometric [68], and RL4CO [19] libraries. Our models were trained on two architectures, each with 10 cores and 10 physical qubits. One architecture features all-to-all connectivity (*A2A*), and the other has a 2×5 mesh grid topology (*Grid*). We assume all-to-all connectivity within cores, as inter-core communication is costlier. We trained models on randomly generated quantum circuits with 30 time slices and 50 or 100 logical qubits, resulting in four trained models: *A2A-50*, *A2A-100*, *Grid-50*, and *Grid-100*.

Each model was trained for 100 epochs using 10,240 randomly generated circuits per epoch. Batch sizes were 128 for 100 qubits and 256 for 50 qubits due to memory limitations. Embedding dimensionality d_E and latent space size d_H were set to 256, with 8 attention heads in all MHA layers, and 3 encoder blocks (b). The learning rate was 10^{-4}, optimized with Adam. Training on a NVIDIA RTX 3060 12GB GPU took approximately 5 hours for 50 qubits and 10 hours for 100 qubits. For a particular circuit, Fig. 8.7b visualizes inter-core state transfers resulting from qubit allocation using the untrained model with random weights. Fig. 8.7c shows the improvement in allocation after policy training for the same circuit.

8.4.2 Iterative black-box optimization approaches

This section presents the results of comparing qubit allocation solutions from iterative derivative-free optimization methods with those from the proposed trained policies. For solving the qubit allocation problem using algorithms like genetic algorithms, we use a suitable solution encoding for algorithm operators (e.g., crossover, mutation) to generate improved solutions based on a fitness function. We avoid encodings that lead to infeasible solutions to maintain sampling efficiency. Instead, we adopt a priority-based encoding, commonly used for problems like the Travelling Salesman Problem [71, 156].

Here, the solution is encoded as a real-valued array $\mathbf{x} \in \mathbb{R}^{T \times P}$, where T is the number of slices and P the total number of physical qubits. During decoding, we initially assume the number of logical qubits equals the number of physical qubits. For each slice, a priority array $\mathbf{x}_t \in \mathbb{R}^P$ is used, and the first C qubits with the highest priority are

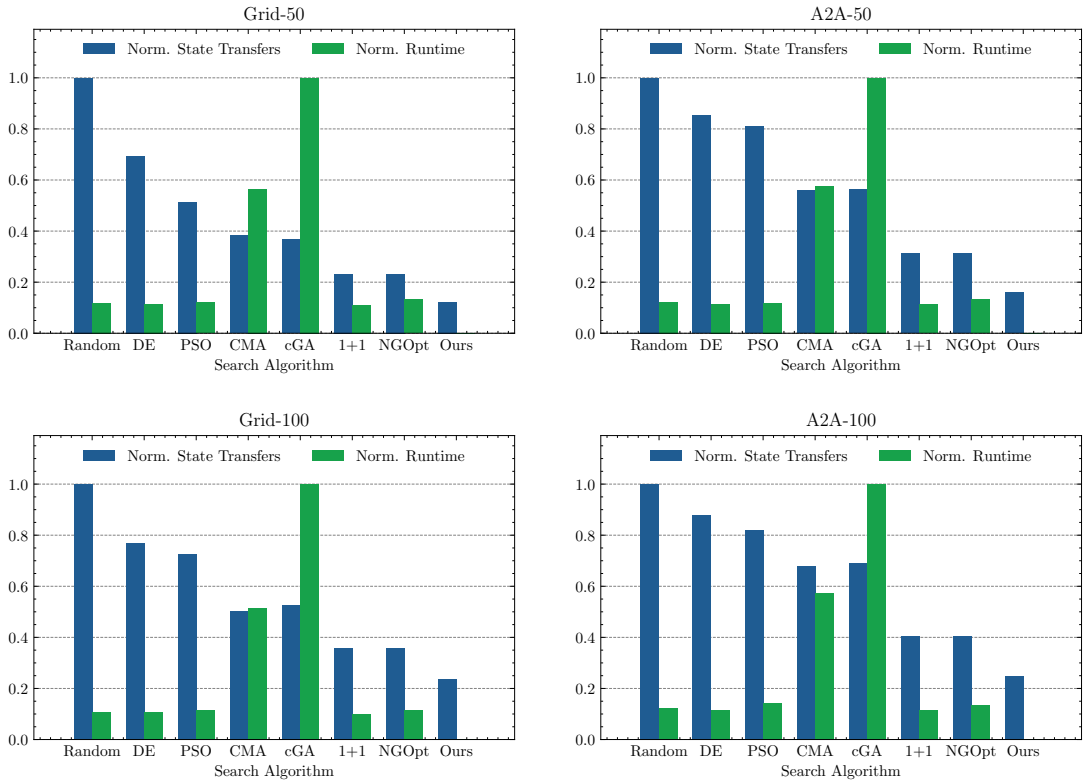


FIGURE 8.8: Inter-core communication count and runtime comparison between the trained policy and black-box iterative optimization approaches on *Grid* and *A2A* architectures.

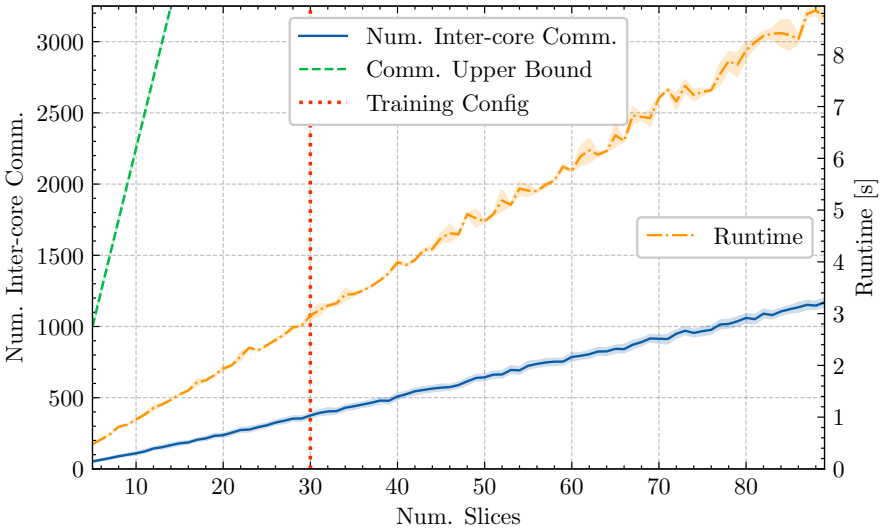


FIGURE 8.9: Number of inter-core communications and allocation runtime when mapping random 50-qubits circuits with a variable number of slices using the *Grid-50* policy.

allocated to the first core, the next C to the second, and so on. The qubit with the highest priority in a gate pair determines the core for its *friend* qubit. Only the first $Q \leq P$ qubits used in the circuit are included in the output solution.

The solution fitness is evaluated using the same reward function as in Eq. (8.19). This encoding allows testing various gradient-free optimization algorithms: Random search, Differential Evolution (DE), Particle Swarm Optimization (PSO), Covariance Matrix Adaptation (CMA), Compact Genetic Algorithm (cGA) [87], One Plus One (1+1) [56], and NGOpt, an adaptive optimization algorithm from Nevergrad. Results comparing these methods with the proposed approach are shown in Fig. 8.8. We tested two random 30-slice circuits with 50 and 100 qubits on both *Grid* and *A2A* topologies using the respective trained policy. The sampling budget for iterative algorithms was set at 10^6 solutions, using default parameters. Search times for iterative methods ranged from 30 minutes to over 4 hours per circuit, while the trained policy returned solutions in seconds. Additionally, the proposed method achieved inter-core communication savings of 33.5% to 48.5% compared to the best baseline.

8.4.3 Generalization capabilities

In Fig. 8.9, we show the performance of the proposed policy, trained on circuits with 50 qubits and 30 slices, tested on circuits with varying slice numbers. The dashed curve represents the upper bound on state transfers for circuits with 5 to 90 slices. The upper

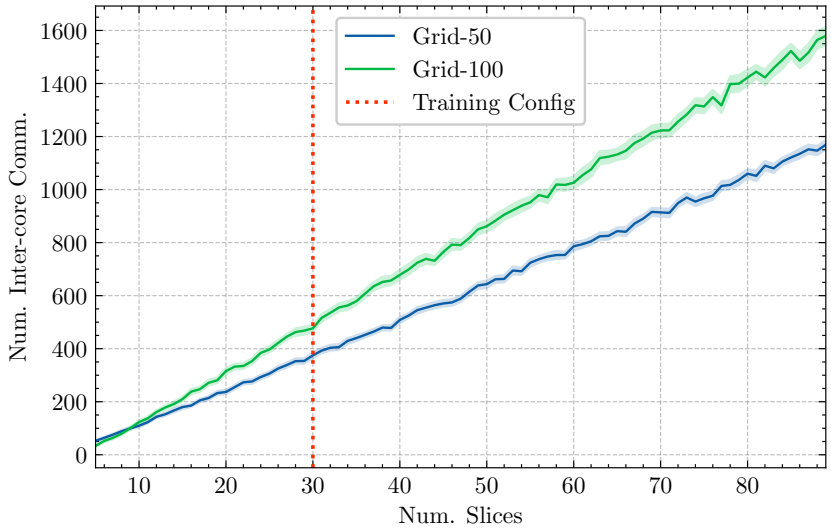


FIGURE 8.10: Comparison of inter-core communications when mapping random 50-qubits circuits using the *Grid-50* and *Grid-100* policies trained on 50-qubit and 100-qubit circuits respectively.

bound $\text{ub}(T, Q)$ is calculated as:

$$\text{ub}(T, Q) = (T - 1) \times Q \times \max \mathbf{D}$$

where T is the number of slices, Q the number of qubits, and $\mathbf{D}$ the distance matrix, considering a 10-core grid topology.

There is no significant degradation in inter-core communications as slice numbers increase, suggesting good generalization for circuits of varying depths. The optimality of the solution for different circuit sizes remains to be assessed, as finding the optimal allocation even for small circuits can take days [84].

Fig. 8.9 also reports runtime in seconds. Runtime grows linearly with circuit depth, up to 8 seconds for 90 slices, due to the slice-wise decoding process. Runtimes refer to single inferences, but on consumer GPUs, multiple allocations can run in parallel (64 allocations for up to 60 slices and 32 for up to 90 slices). Scaling to larger circuits could involve splitting the circuit into smaller chunks, with negligible communication overhead at junctions, calculated as $\text{ub}(K, Q)$ where K is the number of chunks.

The models were trained on random circuits with 50 and 100 qubits, but they can generalize to circuits with fewer qubits. Fig. 8.10 shows results for 50-qubit circuits, where the *Grid-100* policy results in about 24% more state transfers, and this overhead increases with the number of slices. Training on circuits with variable qubit counts could improve generalization.

TABLE 8.1: Trained policy performance for the 2×5 multi-core topology on benchmark circuits.

Benchmark	**Num. Qubits**	**Num. Slices**	**Dual Gate Count**	**Inter-core Comm. Count**
QFT	50	99	1225	1361
	100	197	4950	8918
Quantum Volume	50	50	1226	1892
	100	100	4961	11438
Graph State	50	83	596	827
	100	166	2449	5382
Draper Adder	50	120	925	1049
	100	245	3725	7406
Cuccaro Adder	50	290	336	250
	100	590	686	1782
QNN	50	195	2498	1627
	100	395	9998	10438
Deutsch-Jozsa	50	49	49	125
	100	99	99	570

8.4.4 Comparison with state of the art

In this section we analyze the performance of the trained policies on benchmark quantum circuits. We evaluate the performance for the *Grid* and *A2A* case. For the latter we compare the results with a state-of-the-art technique for multi-core qubit allocation named Fine Grained Partitioning Overall Extreme Exchange (FGP-OEE) and the relaxed version (FGP-rOEE) [13].

These two techniques support fully connected (*A2A*) core topologies only. Table 8.1 shows the inter-core communications for benchmark circuits compiled for the *Grid* architecture (sparse topology) using the trained policies. QFT is the Quantum Fourier Transform without final qubit reordering swaps. Graph State refers to an encoder circuit for a graph with nodes equal to the number of qubits and a random adjacency matrix with 0.5 density. The Drapper and Cuccaro Adders are fixed precision adders for quantum state registers, and QNN represents a Quantum Neural Network circuit. The Deutsch-Jozsa algorithm determines whether a black-box function is constant or balanced. All circuits were optimized and decomposed into two-qubit gates using Qiskit [178], with QNN and Deutsch-Jozsa algorithms from the MQT Benchmark Library [179].

Fig. 8.11 compares the proposed technique with FGP-OEE [13], using the same circuits from Table 8.1 but compiled for the *A2A* multi-core architecture. FGP-OEE

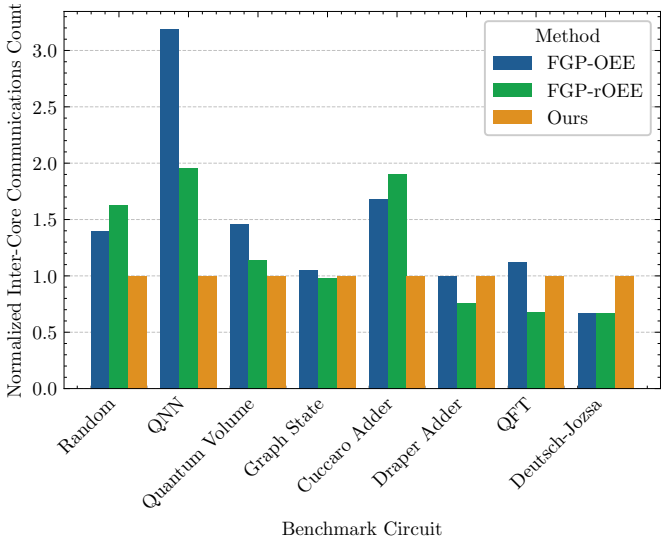


FIGURE 8.11: Number of inter-core communication in *A2A-100* architecture for different benchmark circuits normalized with respect to the output of the proposed technique.

begins by constructing a *lookahead* interaction graph for each time slice with weighted edges, calculated as:

$$w_t(q_i, q_j) = \sum_{t<m\leq T} I(m, q_i, q_j) \cdot 2^{t-m}$$

where $I(m, q_i, q_j) = 1$ if q_i and q_j interact at timestep m. In the second phase, the Overall Extreme Exchange (OEE) graph partitioning algorithm [166] is applied at each timestep.

For a random circuit with 200 slices, the proposed method reduced inter-core communications by 28.26% compared to the baseline. Reductions of 48.82% and 40.53% were measured for QNN and Cuccaro Adders, respectively, while modest improvements were seen for Quantum Volume and Graph State circuits. However, there was a degradation of 32.44% to 47.08% for highly structured circuits like the Draper Adder and QFT, highlighting the challenge of training on random circuits alone. Synthetic circuit dataset generation may help address this [5].

8.5 Conclusion

In our discussion, we formulated the problem assuming communication through state transfers, as often occurs in multi-core architectures. The same approach described in this work, namely a policy that provides qubit assignments for each slice, could be

adapted to single-core cases or, in any case, multi-core scenarios where state movement occurs through swaps. In such cases, the problem becomes more complex, particularly in the calculation of the reward. In the single-core case, for example, it would be necessary to calculate, for each slice, the minimum number of swaps required to transition from one permutation to another, taking into account the system's topology. This problem is known as Token Swapping and is $W[1]$-hard [24].

In future works, a more advanced masking mechanism could could be investigated to deal with cores featuring sparsely connected physical qubits, considering the amount of available edges as capacities instead of free qubits. More advanced policy training algorithms, such as Proximal Policy Optimization (PPO), could potentially lead to better performance by avoiding local minima. Additionally, the policy could be improved by introducing techniques such as relative positional encoding of slices [201], taking into account the intrinsic symmetry of quantum circuits. Regarding the training dataset, circuits resembling real highly structured algorithms could be included rather than considering only randomly generated circuits.

Training attention-based models is computationally intensive but prone to parallel execution [217]. With the availability of increased computational resources, the scalability of the proposed approach to a higher amount of qubits remains to be assessed. Given the promising results about the transformer scalability in context of several thousands of words in the natural language domain, there are reasons to be optimistic about the capability for the proposed attention-based model to scale for thousands of circuit slices.

In conclusion, this work investigated the challenges associated with achieving scalability in quantum computing systems. The imperative need to optimize communication between cores and minimize state transfers while adhering to architectural constraints underscores the complexity of the compilation and mapping of quantum circuits onto physical qubits. Addressing the NP-hard nature of the compilation process, this work has proposed a novel approach employing DRL methods to learn heuristic solutions tailored to a specific multi-core architecture. The experimental evaluations have demonstrated the effectiveness of the proposed method in outperforming baseline approaches, showcasing its ability to reduce inter-core communications and minimize online time-to-solution which we believe makes DRL a promising near-term approach for qubit allocation in modular quantum architectures.

Chapter 9

Heuristic Layout Synthesis in Multi-Core Quantum Systems with Teleport Interconnect

In this chapter, we present a heuristic algorithm for qubit mapping in multi-core quantum architectures aimed at minimizing inter-core state transfers while also optimizing intra-core qubit layouts. The work described in this chapter has been accepted at the *2025 IEEE International Conference on Quantum Computing and Engineering*.

Differently from the previous chapter, in the following discussion we investigate a qubit allocation and routing algorithm that allows to simultaneously optimize intra-core and inter-core qubit layouts. In particular, we focus on multi-core architectures interconnected by means of the teleport protocols described in Section 7.5.1. Unlike conventional SWAP operations used to move logical qubits within cores, teleportation protocols require the consumption of entangled resource states, classical communication channels, and introduce core-local preparation steps considerations that must be carefully managed. Compilers for such distributed quantum systems must not only optimize the placement and routing within individual cores but also orchestrate the timing and resources for inter-core operations.

The SABRE heuristic [135] has previously addressed qubit routing challenges in single-core quantum architectures. This chapter introduces TeleSABRE, a quantum routing framework designed for multi-core quantum processors that extends SABRE heuristics. By utilizing quantum teleportation protocols, TeleSABRE enables efficient qubit transfer and remote gate applications across quantum processing cores. The framework systematically optimizes three critical metrics: the number of qubit swaps, quantum teleportation operations, and remote telegate applications. The core technical contribution of TeleSABRE is its routing algorithm, which employs Dijkstra's shortest path method within the SABRE framework to develop a graph-based approach to quantum resource allocation. This method enables qubit routing across multiple quantum processing cores, addressing the specific connectivity constraints of multi-core quantum

processor architectures. The reference implementation of the proposed method has also been made publicly available: https://github.com/Haimrich/telesabre.

This chapter is organized as follows: Section 9.1 describes the layout synthesis problem in this context. Section 9.2 presents the proposed TeleSABRE heuristic, detailing its algorithmic design and energy model. Section 9.3 evaluates the performance of TeleSABRE against state-of-the-art and near-optimal methods. Section 9.4 discusses limitations and potential future directions. Finally, Section 9.5 concludes the chapter.

9.1 Problem Statement

Consider the scenario depicted in Fig. 9.1, in which a gate must be executed between two logical qubits that are initially located in different cores (Fig. 9.1a). The compiler must determine a sequence of additional operations that either: (1) move the two interacting logical qubits to physical positions that enable execution of the telegate operation, or (2) relocate both logical qubits to the same core and then to physical qubits that allow local gate execution. In Fig. 9.1, we assume the compiler selects the latter approach.

If the compiler's long-term strategy involves performing a teledata operation, the allocation of logical qubits in physical communication qubits must be carefully managed. In Fig. 9.1b, for instance, it becomes necessary to relocate the logical qubit initially assigned to the communication qubit, thereby freeing it for subsequent teleportation operations. Following this, as required by the teledata protocol (Fig. 7.7), a Bell state can be distributed to the two communication qubits (Fig. 9.1c). The protocol then proceeds with the preprocessing step (Fig. 9.1d), measurement and classical communication, i.e., the transmission of two classical bits, also known as the "phone call" (Fig. 9.1e), and conditional gate execution (Fig. 9.1f).

It is important to note that traditional SWAP insertions needed in single-core architectures must also be considered to enable gate execution in the destination core. Consequently, in the step shown in Fig. 9.1f, a SWAP operation is applied to facilitate local gate execution in the final step (Fig. 9.1g).

Similar considerations apply to telegate execution. Furthermore, since a circuit typically consists of multiple gates, optimizing the number of qubit movement operations (SWAPs, telegates, and teleports) requires comprehensive analysis to select operations that benefit the greatest number of gates to be executed. When choosing between telegate and teleport operations, the compiler should consider the frequency of future interactions involving logical qubits currently allocated in each core. For more complex multi-core architectures, inter-core routing becomes necessary to determine the optimal teleportation path between cores. All these factors must be balanced while also addressing intra-core movements required for local gate execution and for freeing communication qubits needed in teleportation protocols.

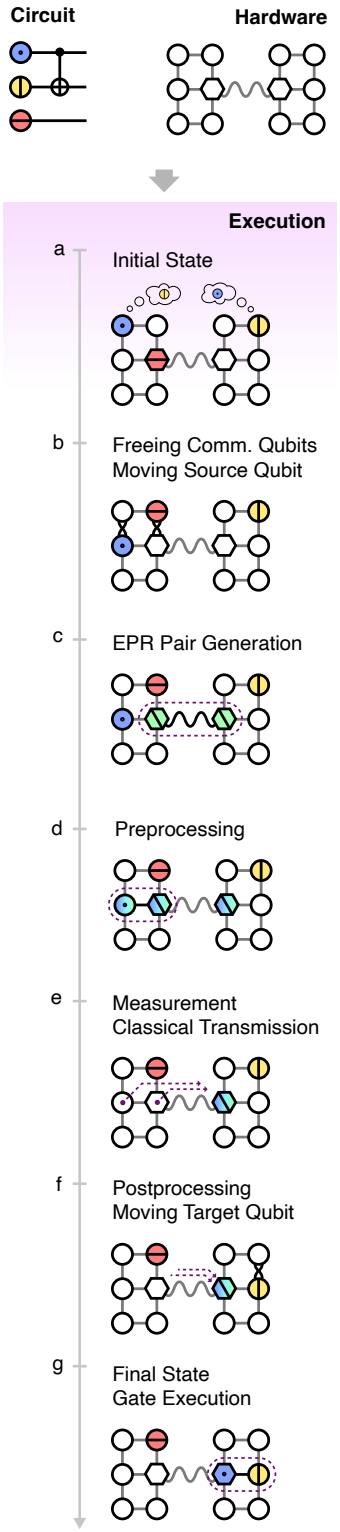


FIGURE 9.1: Example multi-core quantum layout synthesis scenario.

9.2 Methodology

In this section, we introduce the TeleSABRE heuristic algorithm. In Section 9.2.1, we describe the top-level flow of the proposed algorithm adapted from SABRE [135]. Then, in Section 9.2.2 we provide details about the energy calculation and we distinguish between local energy calculation (Sec. 9.2.3) and remote energy calculation (Sec. 9.2.4). Section 9.2.5 describes how candidate operations are selected at each iteration. Finally, we describe the possibility to further optimize the starting layout in Sec. 9.2.6 and we highlight the algorithmic complexity in Sec. 9.2.7.

9.2.1 Algorithm Overview

Algorithm 4 SABRE-based Search Algorithm

Input: Circuit DAG $\mathcal{G}$, Architecture $\mathcal{A}$, *seed*
Output: Routed Circuit, Final Layout λ_f

$i \leftarrow 0$
$\lambda_0 \leftarrow$ `initialLayout(`$\mathcal{A}$`,` *seed*`)`
usage $\leftarrow$ `onesArray(`$\mathcal{A}$.*num_qubits*`)`
while *front*($\mathcal{G}$) $\neq \emptyset$ **do**
  executable_gates = $\emptyset$
  for *gate* $\in$ *front*($\mathcal{G}$) **do**
    if *canRunGate*($\mathcal{A}$,λ_i,*gate*) **then**
      executable_gates.`append(`*gate*`)`
  if *executable_gates* $\neq \emptyset$ **then**
    for *gate* $\in$ *executable_gates* **do**
      $\mathcal{G}$.`removeNode(`*gate*`)`
    $\lambda_{i+1} = \lambda_i$
  else
    scores = `[]`
    candidate_ops = `ObtainCandidateOps(`$\mathcal{A}, \mathcal{G}, \lambda_i$`)`
    for *op* $\in$ *candidate_ops* **do**
      λ_{op} = `applyOperation(`λ_i, op`)`
      energy = `calculateEnergy(`$\mathcal{A}, \mathcal{G}, \lambda_i$`)`
      scores[*op*] = *energy* $*$ max{*usage*[p] | $p \in op$}
    op $\leftarrow$ `sample(`{*op* | *scores*[*op*] = min *scores*}`)`
    λ_{i+1} = `applyOperation(`λ_i, op`)`
    usage $\leftarrow$ `updateUsagePenalties(`*usage*, *op*`)`
  $i \leftarrow i + 1$

The primary flow of the algorithm, based on SABRE [135], is outlined in Algorithm 4. TeleSABRE takes as input a circuit directed acyclic graph (DAG), denoted as $\mathcal{G}$ (Fig. 7.4b), and a multi-core quantum architecture, $\mathcal{A}$, and produces a sequence of layout-feasible gate executions along with additional SWAP and teleportation operations.

The algorithm begins by generating an initial layout, which assigns logical qubits to physical qubits as described in Sec. 9.2.6. Subsequently, the main loop of the algorithm is initiated. In each iteration, the algorithm considers the front layer of the remaining DAG, which consists of the remaining gates eligible for execution, i.e., the subset of gates whose dependencies have been satisfied. The gates in this frontier that are executable given the current qubit layout are scheduled for execution and subsequently removed from the DAG. If no gates are available for execution, the algorithm considers introducing a movement operation to facilitate progress.

In the original SABRE algorithm, the only available movement operation is the SWAP operation. In contrast, TeleSABRE extends this approach by incorporating additional movement operations, specifically logical qubit teleportation (teledata) and gate teleportation (telegate). Each candidate operation is evaluated based on a heuristic score function, which we refer to as energy in this work. The algorithm virtually applies each candidate operation to the current state and computes the energy of the resulting layout. The operation that yields the resulting layout with lowest energy is selected for execution. In cases where multiple candidates result in the same energy, one is randomly selected.

To ensure load balancing and minimize circuit depth, the algorithm incorporates a physical qubit usage factor into the energy calculation called decay. This factor is updated each time an operation is applied to a physical qubit, promoting a homogeneous distribution of movement operation across the architecture increasing parallelization and reducing compiled circuit depth.

9.2.2 Energy Calculation

In the original SABRE [135, 242] implementation, the energy of a given state is computed as follows:

$$\text{energy} = \frac{1}{|F|} \sum_{g \in F} \text{energy}_\lambda^G(g) + \underbrace{\frac{k}{|H|} \sum_{g \in H} \text{energy}_\lambda^G(g)}_{\text{lookahead component}} \tag{9.1}$$

where $F = \texttt{front}(\mathcal{G})$ represents the current frontier of the remaining circuit DAG, consisting of gates whose dependencies have been satisfied and are ready for execution. The set H, referred to as the *extended set*, serves as a lookahead horizon, encompassing a subset of gates from the remaining circuit DAG that are scheduled for execution in subsequent steps. The parameter k denotes the lookahead factor, which adjusts the relative importance of future gates. The function $\text{energy}_\lambda^G(g)$ represents the energy associated to the gate g given the current qubit layout λ.

9.2.3 Local Energy Calculation

In the original SABRE algorithm, for single core quantum architectures, the energy associated to each two-qubit gate g acting on logical qubits $q_{g,1}$ and $q_{g,2}$ is calculated as:

$$\text{energy}_{\lambda}^{G}(g) = \text{dist}_{\lambda}(g) = \mathbf{D}[\text{phys}_{\lambda}(q_{g,1})][\text{phys}_{\lambda}(q_{g,2})] \tag{9.2}$$

where $\text{phys}_{\lambda}(q_{g,1})$ and $\text{phys}_{\lambda}(q_{g,2})$ are physical qubits hosting $q_{g,1}$ and $q_{g,2}$ respectively according to layout λ, and $\mathbf{D}$ is the hop distance matrix for physical qubits in the quantum architecture.

The application of a SWAP operation can either increase or decrease this potential energy: it increases if it results in a greater overall distance between pairs of interacting qubits (i.e., those involved in the same gate), and conversely, it decreases if it brings these qubits closer together.

The distance matrix can be precomputed in $O(N^3)$ time using the Floyd-Warshall algorithm for a given coupling map of a monolithic quantum processor with N physical qubits. This precomputed matrix is then utilized throughout the execution of SABRE to determine the distance between interacting logical qubits.

9.2.4 Remote Energy Calculation

Extending this approach to a teleportation-interconnected multi-core architecture, however, is non-trivial. As described in Section 7.5.1, executing a teleportation protocol is a multi-step process. Simply considering the distance between interacting qubits, even if they reside in different cores, is insufficient. A teleportation operation requires several intermediate steps: first, two communication qubits must be entangled; then, the source qubit must be entangled with the mediator physical qubit. Only after these steps can the quantum state be transferred to the target communication qubit. From a layout perspective, a teleportation operation is admissible only if the following conditions are met:

1. The two communication qubits must be *free*, i.e., no logical qubits involved in the circuit should be mapped onto them.

2. The quantum state to be teleported must reside in a physical qubit adjacent to the mediator communication qubit.

3. The destination core must have sufficient available capacity.

Thus, the extended energy calculation in TeleSABRE must account not only for the distance between logical qubits but also for the progress toward teleportation operations that can reduce this distance. Additional SWAP operations may be necessary to free the communication qubits required for teleportation. Moreover, an inter-core routing

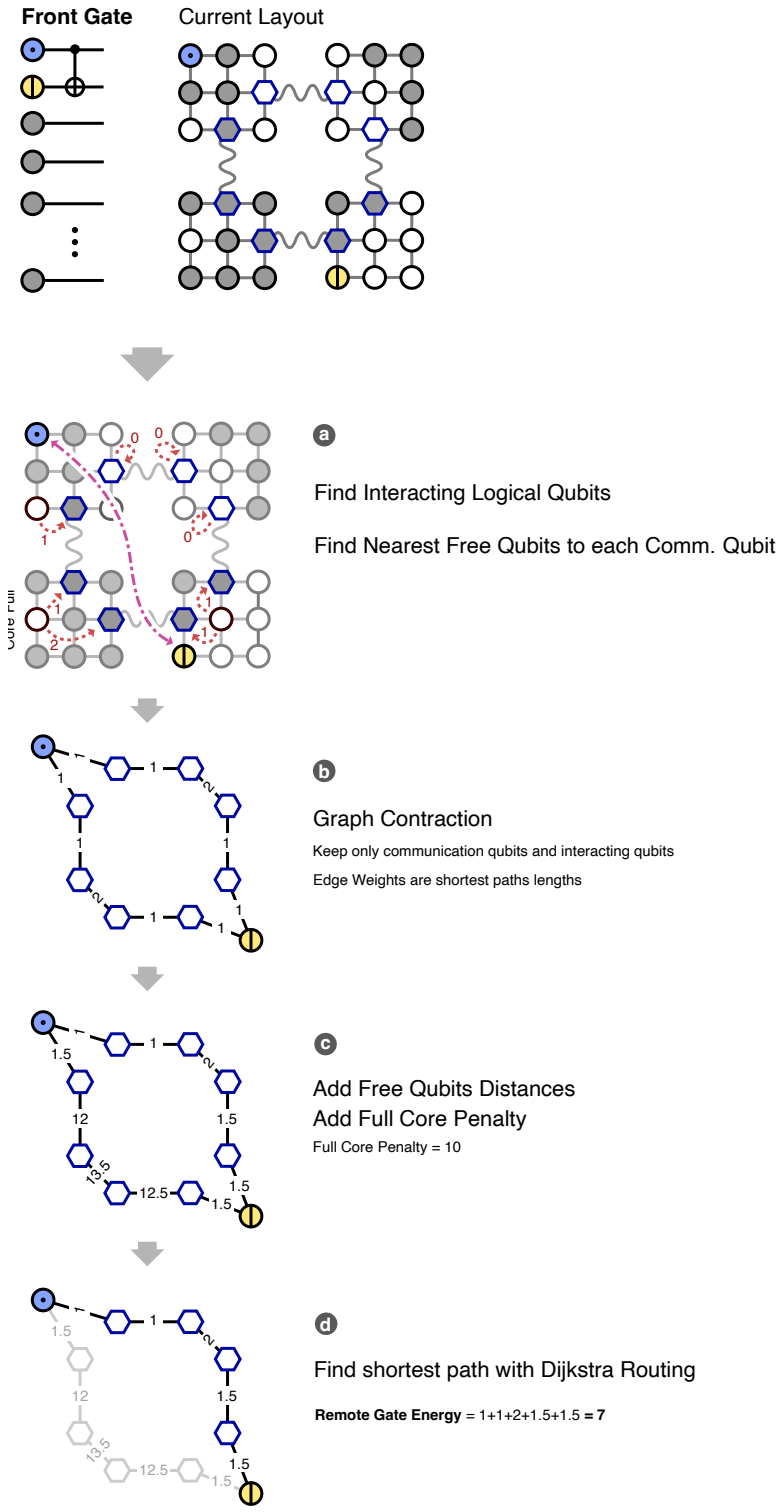


FIGURE 9.2: Contracted graph generation and remote gate energy calculation.

mechanism is required to establish a viable path between the cores of the two interacting qubits. This routing process must consider the current capacity of each core, ensuring that an inter-core path does not pass through a full core. A core is considered full if it has only one free physical qubit remaining.

To prevent deadlock situations, layouts in which a core has no free physical qubits are disallowed, as such a configuration would prevent freeing the communication qubit via SWAP operations, thereby making qubit transfer out of the core impossible. If multiple communication qubits per core are available, the inter-core routing strategy should also account for their relative distances from the logical qubits involved in the gate operation. Additionally, certain communication qubits, despite being physically closer, may still be occupied by a logical qubit, making entanglement infeasible until further SWAP operations are performed to free them.

For TeleSABRE to be effective, the routing process must incorporate all these factors, and the energy function should be designed to reflect these considerations. We achieve this by performing routing on a contracted graph, where edge weights are adjusted to account for the swap (hop) distance (precomputed in $\mathbf{D}$) between physical qubits, the proximity of the nearest free qubit to the communication qubits, and penalties associated with cores that have depleted capacity. The process of constructing the contracted graph and performing routing is illustrated in Fig. 9.2. The procedure begins by identifying the physical qubits corresponding to the two disjoint logical qubits involved in the same gate operation, nearest free physical qubits to each communication qubit and their distance as highlighted by red dashed arrows in Fig. 9.2a. Next, the contracted graph is constructed by retaining only the communication qubits and the two qubits involved in front gate under consideration, with edge weights set to the shortest path length between them (Fig. 9.2b). For edges that connect the physical qubits hosting the interacting logical qubits to the communication qubits, the edge weight is adjusted by subtracting one and then taking the absolute value. This ensures that the weight becomes zero only when the interacting qubits are adjacent to the communication qubits, without overlapping. To account for the accessibility of communication qubits, half of the hop distance to the nearest free qubit is added to the weights of the two edges incident to each communication qubit (Fig. 9.2c). This adjustment reflects the number of SWAP operations required to free a communication qubit if it is part of the shortest path. Furthermore, if a core has exhausted its capacity (i.e., fewer than two free qubits remain), an additional penalty is incorporated into the edge weights of communication qubits associated with that core.

Finally, as shown in Fig. 9.2d, Dijkstra's algorithm [54] is applied to determine the shortest path length (i.e., sum of the weights of the shortest path edges) $\text{routing}_\lambda(g)$ between the two disjoint gate qubits $(q_{g,1}, q_{g,2})$, which represents the minimum number of operations required to execute the gate. When multiple gates require inter-core routing in the same TeleSABRE iteration, traffic on inter-core links connecting two communication qubits on different core, can be considered by adding a traffic coefficient to

the corresponding contracted graph edges before performing the routing for the next front gate. The example shown in Fig. 9.2 shows a trivial situation in which finding the shortest path is naive as one of the two intermediate cores is full and each core has only two communication qubits, hence no intra-core routing between communication qubits is needed.

Hence, the TeleSABRE total state energy can be still defined as in Eq. (9.1), but the definition of the gate energy is updated as follows:

$$\text{energy}_{\lambda}^{G}(g) = \begin{cases} \text{dist}_{\lambda}(g), & \text{if } \text{core}_{\lambda}(q_{g,1}) = \text{core}_{\lambda}(q_{g,2}) \\ \text{routing}_{\lambda}(g), & \text{otherwise} \end{cases}$$

9.2.5 Candidate Operations Selection

One of the key strengths of the SABRE algorithm is its ability to avoid evaluating all possible SWAP operations that could be applied to the layout. Instead, it restricts the set of candidate SWAPs at each iteration to those involving physical qubits where the logical qubits of gates in the current front are currently allocated.

We adopt a similar approach but introduce modifications to accommodate logical qubit teleportation and gate teleportation. Specifically, the set of candidate SWAPs must also include those involving the nearest free qubits to communication qubits that participate in the routing plans determined by Dijkstra's algorithm for gates in the front layer where the two logical qubits reside in different cores. In addition to SWAP operations, the set of candidate operations must also incorporate feasible teleportation and telegate operations based on the current layout.

9.2.6 Initial Assignment

The initial layout is generated by prioritizing the placement of interacting qubits from the first frontier of the circuit's DAG within the same cores [61], while intra-core positioning is randomly assigned. This initial configuration can serve as a starting point for further initial layout optimization, following an approach analogous to the original SABRE methodology [135]. To obtain an optimized initial layout, the layout synthesis process can be carried out in three phases: first, a forward TeleSABRE pass from an initial layout is performed, after which the resulting final layout is utilized for a second backward TeleSABRE pass on the reversed quantum circuit. The final layout obtained from this backward pass then serves as the initial layout for the definitive forward pass, completing the bidirectional optimization procedure.

9.2.7 Algorithmic Complexity

The original SABRE algorithm achieves a significant reduction in computational complexity by restricting the search space for SWAP operations, reducing it from $O(\exp(N))$

to $O(N)$. In the worst case, where all qubits are involved in the front layer, this approach remains scalable. Although additional SWAP operations may be required to bring qubits together, the maximum number of SWAPs needed for a two-qubit gate is bounded by the diameter of the chip coupling graph, which is $O(\sqrt{N})$ for a two-dimensional layout. Furthermore, evaluating the energy function is $O(N)$[1]. As a result, the overall complexity for each two-qubit gate is reduced to at most $O(N^{2.5})$ [135].

In TeleSABRE, when two qubits involved in a gate reside in different cores, routing decisions require executing Dijkstra's algorithm on a contracted graph that consists of communication qubits. The number of communication qubits is proportional to the number of cores, C. Since Dijkstra's algorithm runs in $O(V \log V + E)$ time for a graph with V nodes and E edges (using a priority queue implementation), and the number of communication qubits is $O(C)$, the additional computational overhead for inter-core routing is $O(C \log C)$. In the worst-case scenario, where every two-qubit gate requires inter-core routing, the total complexity per two-qubit gate increases to $O(N^{2.5} \cdot C \log C)$. This result indicates that while our approach introduces additional computational overhead due to inter-core routing, it remains scalable when C is moderate compared to N. In this analysis, we assumed that finding the nearest free qubits to the communication qubits is $O(1)$ because for each communication qubit a sparse bucket queue can be used to keep track of the nearest free qubits as swaps and teleport are applied on them. Optimized implementations considering the calculation of the energy difference to greatly reduce complexity could be investigated in future research [242].

9.3 Experiments

In this section we evaluate the proposed methodology, comparing it against state-of-the-art methods and near-optimal methods and evaluating the impact of optimizing the initial layout.

9.3.1 Experimental Setup

We implemented the proposed methodology using Python. To evaluate the performance of TeleSABRE, we considered the multi-core architectures illustrated in Fig. 9.3, with the number of physical qubits ranging from 8 to 96. Our evaluation utilized a diverse set of benchmark circuits from the Qiskit framework [104] and MQTBench [179], including: Random circuit, Greenberger–Horne–Zeilinger state preparation (GHZ), Graph State preparation, Portfolio Optimization with Quantum Approximate Optimization Algorithm (Portfolio QAOA), Amplitude Estimation (AE), Quantum Fourier Transform (QFT), Quantum Neural Network (QNN), Draper Adder, and Cuccaro Adder. All circuits were optimized, transpiled, and decomposed using Qiskit to target a native gate

[1] This can be reduced to $O(1)$ with an optimized implementation [242]

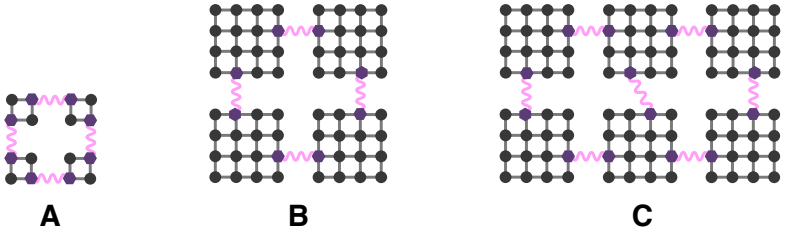


FIGURE 9.3: Architectures considered in experiments

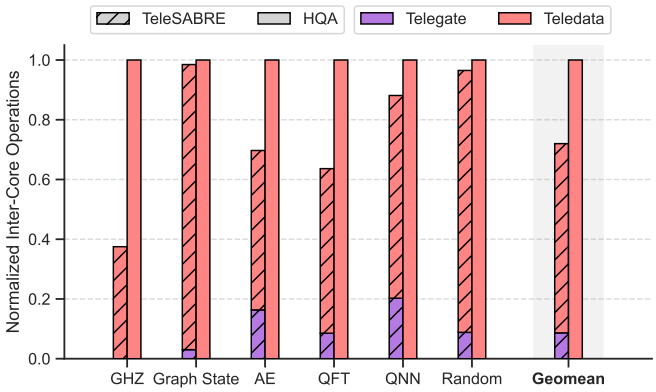


FIGURE 9.4: Inter-core operation count comparison against Hungarian Qubit Assignement [61] for benchmark circuits with 64 qubits on architecture C.

set consisting of rotation around z-axis (RZ), square root of NOT (SX), bit-flip (X), and controlled bit-flip (CX) operations. The experiments were conducted with varying numbers of logical qubits (8, 26, or 64). Number of introduced teleport, telegate and SWAP operations and compiled circuit depth were considered as evaluation metric.

9.3.2 Comparison against state of the art

In this section we compare against a state-of-the-art method for inter-core state transfer minimization in multi-core quantum architectures, namely Hungarian Qubit Assignment (HQA) [61]. We consider the same initial qubit assignment in cores for both the methods and compare the amount of introduced inter-core operations, either state transfer (teledata) or remote gates (telegate). We consider architecture C with 96 physical qubits (Fig. 9.3c) and benchmark circuits with 64 qubits. The results are shown in Fig. 9.4. Except for the Graph State preparation circuit and the Random circuit, TeleSABRE achieves reductions in terms of inter-core operations ranging from 11.8% to 62.5%. The geometric mean of reduction across all the benchmarks circuit is 28%.

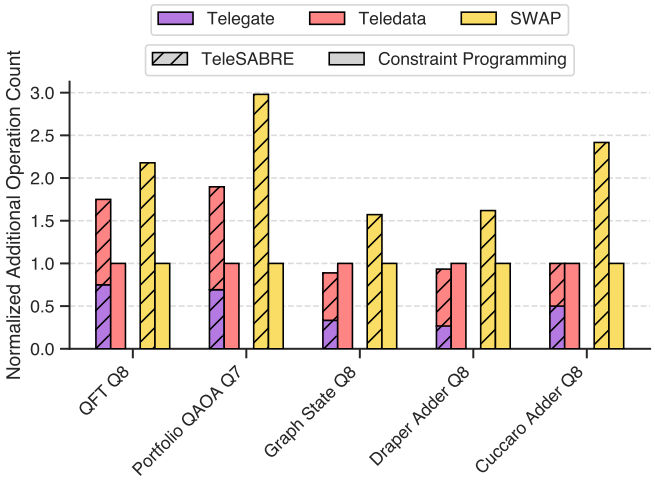


FIGURE 9.5: Operations introduced by layout synthesis compared to near-optimal constraint programming solutions for circuits up to 8 qubits on architecture A.

9.3.3 Comparison against near-optimal

In this section, we evaluate the optimality of solutions generated by TeleSABRE. We employ a block-based constraint programming formulation for the layout synthesis problem in multi-core quantum systems to find near-optimal solutions. Our approach extends TB-OLSQ2 [138] by accounting for intra-core swaps, inter-core teleports, and communication qubit logistics. We implement our model using OR-Tools [170]. The optimization follows a hierarchical objective: first minimizing transitions [138], then teleports, and finally swaps. As solving the layout synthesis problem to optimality is NP-complete [25], we are able to find near-optimal solutions only for small circuit instances and a small architecture, namely architecture A in Fig. 9.3. Due to the stochasticity of the proposed algorithm, we run TeleSABRE 3 times and consider the best solution in terms of the count of inter-core operations. Fig. 9.5 shows the amount of inter-core and intra-core operations introduced by the two compilation methods. TeleSABRE finds solution with up to three times the near-optimal amount of swaps and two times the near-optimal amount of teleports. On the smaller A architecture we also notice an increase in the amount of telegates with respect to teledata operations. In fact, in bigger architectures more hops (teledata) are necessary on average to obtain two interacting qubits on adjacent cores.

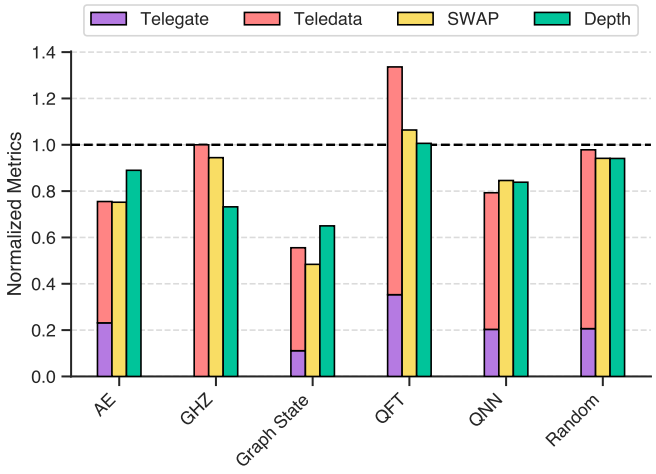


FIGURE 9.6: Normalized metrics comparison with respect to non-optimized initial layout.

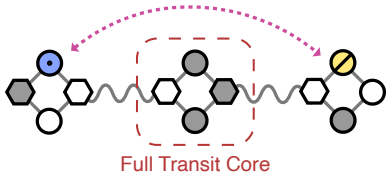


FIGURE 9.7: Example deadlock scenario.

9.3.4 Initial Layout Optimization

As described in Section 9.2.6, it is also possible to further optimize the initial layout similarly to the original SABRE implementation. In this section, we evaluate the improvements in terms of different metrics with respect to the selection of the initial layout. We considered architecture B (Fig. 9.3b) and 25 logical qubit circuits. Except for the QFT benchmark in which running the two extra forward-backward passes result in a worse initial layout, for all the other benchmarks the improvements are made up to 44% in terms of inter-core operations, up to 51% in terms of additional intra-core SWAP operations and up to 35% in terms of circuit depth.

9.4 Limitations and future work

The proposed approach provides a heuristic baseline for holistic quantum compilation in multi-core architectures with teleport interconnects. In this section, we describe limitations that could be addressed in future work.

A first limitation concerns the possibility of encountering deadlocks during the layout synthesis process. For instance, consider the scenario in Fig. 9.7, where the Dijkstra routing phase fails to find a feasible teleport path between two cores because the only possible path contains a core with depleted capacity. In this case, the only viable operation is to perform a teleport to free a spot in the intermediate core. Deadlock situations are also present in the original SABRE algorithm and have been solved through a safety valve mechanism [242]; a similar approach could be explored in future research for TeleSABRE.

Another notable limitation is the absence of mechanisms that exploit the full range of possibilities offered by entanglement. In TeleSABRE, entangled pairs are not explicitly tracked; instead, the system only monitors whether it is possible to generate them as needed for teleportation between directly connected cores. In practice, however, Bell-state qubits can be relocated after their initial creation from the physical communication qubits on which they are prepared. Moreover, techniques such as entanglement swapping enable teleportation between cores that are not directly linked, as long as a connecting path exists in the network topology [27, 66]. Additionally, advanced protocols utilizing primitives such as *cat-entanglement* and *cat-disentanglement* [234] may be employed to perform multiple remote controlled phase-flip gates acting on a shared qubit, requiring only a single EPR pair [67, 93].

Finally, similar to the first version of SABRE [135], TeleSABRE was implemented in Python. Further effort should be devoted to developing an efficient implementation, as was done for the original SABRE algorithm in [242]. Furthermore, the Dijkstra algorithm could be replaced with the A* algorithm, using the number of cores with depleted capacity along a path as a heuristic to quickly prioritize paths with available resources.

9.5 Conclusion

In this chapter, we introduced TeleSABRE, an extension of the SABRE algorithm that addresses the unique challenges of layout synthesis in multi-core quantum architectures featuring teleportation-based interconnects. By integrating both intra-core SWAP operations and inter-core teleportation protocols, including teledata and telegate primitives, TeleSABRE offers a unified framework that holistically optimizes intra-core qubit movement, gate scheduling, and inter-core communication. Our results demonstrate that TeleSABRE can also significantly reduce the number of inter-core operations across a range of benchmarks while taking into account local operations needed for teleportation protocols.

Future research directions include developing advanced deadlock avoidance mechanisms, integrating entanglement management techniques such as entanglement swapping,

and exploring alternative heuristic and learning-based formulations for the energy function. Scaling the implementation to support larger quantum systems and more complex architectures remains an important objective. It is also critical to examine how algorithm parameters influence the four critical output metrics, i.e., amount of teledata, telegate, SWAP operations and circuit depth, and assess their impact on circuit fidelity [95]. Additionally, the tradeoff between SWAP intra-core operations and Telegate inter-core operations across different architectural topologies and their impact on fidelity should be investigated.

Furthermore, the publicly available implementation of TeleSABRE may serve as a practical baseline for further research in quantum compilation for modular architectures.

Chapter 10

Conclusion

The journey of this thesis began with Moore's Law and Dennard scaling, two foundational pillars whose erosion has redefined the landscape of computing. As device-level scaling has slowed, and power and thermal walls have arisen, the semiconductor industry has been compelled to look beyond traditional means for performance and efficiency gains. This has precipitated a profound shift: from homogeneous, general-purpose architectures to heterogeneous systems embracing specialization, and from purely classical platforms to the nascent era of quantum computing.

At the heart of this transformation lies the unprecedented rise of artificial intelligence, particularly deep neural networks, which now permeate domains ranging from computer vision and natural language processing to autonomous systems and scientific discovery. The computational and energy demands for both training and inference of these models have grown exponentially, challenging the capabilities of conventional hardware and driving the need for tailored domain-specific accelerators. In parallel, the emergence of quantum computing introduces fundamentally new paradigms and constraints, promising algorithmic breakthroughs for certain classes of problems but bringing forth daunting challenges in hardware reliability, compilation, and mapping.

The contributions of this thesis are situated at the confluence of these trends. We have developed and evaluated tools and methodologies for efficient design space exploration, mapping, and scheduling across both AI accelerators and quantum architectures, recognizing that future computing platforms will be defined not by a single technology but by their ability to orchestrate diverse hardware resources to meet increasingly complex workload requirements.

Summary of Contributions

This dissertation presented:

- **LEMON**: A mathematical programming-based mapping tool for DNN layers on spatial accelerators, balancing memory access costs and bandwidth constraints to achieve near-optimal mappings. LEMON's efficient formulation enables rapid

evaluation of mapping strategies, laying the groundwork for future extensions that address data sparsity, imperfect factorization, uneven mappings, and cross-layer data movement.

- **MOHaM**: A genetic algorithm-based framework for the co-optimization of architecture, mapping, and scheduling in multi-accelerator, multi-DNN systems. By efficiently traversing the vast design space, MOHaM identifies Pareto-optimal solutions and demonstrates potential for practical use in real-world scenarios like ADAS and AR/VR systems. The framework's flexibility positions it as a foundation for further advances in scheduling space exploration and rapid cost modeling.
- **RELMAS and GRAMAS**: Reinforcement learning and data-driven approaches to online scheduling and dataflow optimization for heterogeneous multi-accelerator systems, including specialized graph neural network inference. These methods demonstrate that learning-based approaches can significantly improve resource utilization, job completion times, and adaptability across varying workloads, and open new research directions in state encoding, fairness, and SLA-aware scheduling.
- **Quantum Mapping and Routing**: Reinforcement learning and heuristic techniques for scalable qubit mapping and routing in modular quantum systems, addressing both intra-core and inter-core communication. The DRL-based mapping approach shows promising results in reducing inter-core state transfers and adapting to architectural constraints, while the teleportation-aware routing heuristic (TeleSABRE) holistically optimizes qubit movement and communication. Both lines of work provide practical baselines and identify open avenues for integrating advanced learning, masking mechanisms, deadlock avoidance, and entanglement management.

Implications and Outlook

Collectively, the tools and frameworks developed in this work contribute to automated design space exploration, mapping, and scheduling for heterogeneous and emerging architectures, and the results emphasize several interrelated themes. Co-design is essential: achieving Pareto-optimal trade-offs in performance and efficiency for contemporary and future systems requires simultaneous consideration of hardware and workload characteristics, since isolated or sequential design flows cannot keep pace with the way application demands and device constraints are becoming intertwined. Learning-driven optimization has likewise proven valuable; reinforcement learning and data-driven predictors effectively navigate complex, high-dimensional scheduling and mapping spaces, and as workloads and hardware platforms grow in scale and diversity, these adaptive techniques will become increasingly critical. Finally, the work highlights the close methodological

ties between classical and quantum domains: many architectural and compilation challenges encountered in AI accelerators reappear in new forms in quantum computing, and techniques from classical domain-specific accelerators—particularly mapping and scheduling under resource constraints—provide a useful foundation for addressing the unique demands of quantum architectures.

Future Directions

The contributions of this thesis open multiple promising research directions. Extending DSE tools to richer design and mapping spaces that incorporate sparsity, non-uniform mappings, cross-layer data movement, and more realistic workload traces will broaden applicability and improve fidelity. Integrating tenant-aware, SLA-driven, and fairness-aware mechanisms into online scheduling policies, potentially leveraging advanced neural architectures and dynamic reward shaping, will be vital for multi-tenant systems. For quantum architectures, further investigation into sophisticated masking strategies, positional encodings, and learning algorithms (for example, PPO and attention-based models), together with the integration of entanglement management and deadlock-avoidance heuristics, will be crucial to scale learning-based mapping to larger devices. More broadly, as computing platforms become ever more heterogeneous, there is a growing need for unified cross-domain frameworks that seamlessly bridge classical and quantum components, leveraging shared abstractions and optimization strategies to enable coherent, co-designed systems.

Final Remarks

In summary, this thesis has contributed new methodologies and tools for the efficient co-design and deployment of domain-specific accelerators and quantum architectures. As the boundaries between classical and quantum, hardware and software, and architecture and application continue to blur, we believe that such cross-layer, data-driven, and adaptable approaches will be central to unlocking the next era of computing performance, efficiency, sustainability and capability.

Acronyms

AI	Artificial Intelligence.
ALU	Arithmetic Logic Unit.
AM	Application Model.
ANN	Artificial Neural Network.
AR	Augmented Reality.
ASIC	Application-Specific Integrated Circuit.
BW	Bandwidth.
CMOS	Complementary Metal-Oxide-Semiconductor.
CNN	Convolutional Neural Network.
CPU	Central Processing Unit.
CSR	Compressed Row Storage or Control/Status Register.
CV	Computer Vision.
CVRP	Capacitated Vehicle Routing Problem.
CX	Controlled-NOT quantum gate.
DAG	Directed Acyclic Graph.
DL	Deep Learning.
DNN	Deep Neural Network.
DRAM	Dynamic Random Access Memory.
DRL	Deep Reinforcement Learning.
DSA	Domain-Specific Accelerator.
DSE	Design Space Exploration.
DSP	Digital Signal Processor.
EDP	Energy-Delay Product.
EPR	Einstein-Podolsky-Rosen.
FC	Fully Connected.

FLOPS	Floating Point Operations Per Second.
FPGA	Field Programmable Gate Array.
FTQC	Fault-Tolerant Quantum Computing.
GA	Genetic Algorithm.
GAT	Graph Attention Network.
GCN	Graph Convolutional Network.
GEMM	General Matrix Multiply.
GIN	Graph Isomorphism Network.
GNN	Graph Neural Network.
GPU	Graphic Processing Unit.
GRS	Ground-referenced signaling.
HBM	High Bandwidth Memory.
HDA	Heterogenous Dataflow Accelerator.
ILP	Integer Linear Programming.
IoT	Internet-of-Things.
ISP	Image Signal Processor.
LSTM	Long-Short-Term Memory.
MAC	Multiply-and-Accumulate operation or unit.
MAS	Multi-Accelerator System.
MCM	Multi-Chip Module.
MDP	Markov Decision Process.
MHA	Multi-head Attention.
MI	Memory Interface.
MIP	Mixed-Integer Programming.
ML	Machine Learning.
MLP	Multi-Layer Perceptron.
MOEA	Multi-Objective Evolutionary Algorithm.
NISQ	Noisy Intermediate-Scale Quantum.
NLP	Natural Language Processing.
NoC	Network-on-Chip.
NoP	Network-on-Package.
NPU	Neural Processing Unit.

PE	Processing Element.
PPO	Proximal Policy Optimization.
PPU	Post-processing Unit.
QEC	Quantum Error Correction.
QoS	Quality of Service.
QPU	Quantum Processing Unit.
QUBO	Quadratic Unconstrained Binary Optimization.
ReLU	Recified Linear Unit activation function.
RL	Reinforcement Learning.
RNN	Recurrent Neural Network.
RTL	Register Transfer Level.
SA	Sub-Accelerator.
SAI	Sub-Accelerator Instance.
SAT	Sub-Accelerator Template.
SIMD	"Single instruction, multiple data" computer architecture.
SLA	Service Level Agreement.
SLI	Service-Level Indicator.
SLO	Service-Level Objective.
SMT	Satisfiability Modulo Theory.
SoC	System-on-Chip.
SpMM	Sparse Matrix Multiply.
SRAM	Static Random Access Memory.
TSP	Traveling Salesman Problem.
VR	Virtual Reality.

List of Figures

List of Tables

Bibliography

[1] S. Abadal, A. Jain, R. Guirado, J. López-Alonso, and E. Alarcón, "Computing graph neural networks: A survey from algorithms to accelerators", *ACM Comput. Surv.*, vol. 54, no. 9, 2021, ISSN: 0360-0300. DOI: 10.1145/3477141. [Online]. Available: https://doi.org/10.1145/3477141.

[2] J. Achiam, S. Adler, S. Agarwal, L. Ahmad, I. Akkaya, F. L. Aleman, D. Almeida, J. Altenschmidt, S. Altman, S. Anadkat, *et al.*, "Gpt-4 technical report", *arXiv preprint arXiv:2303.08774*, 2023.

[3] L. Ale, N. Zhang, H. Wu, D. Chen, and T. Han, "Online proactive caching in mobile edge computing using bidirectional deep recurrent neural network", *IEEE Internet of Things Journal*, vol. 6, no. 3, pp. 5520–5530, 2019. DOI: 10.1109/JIOT.2019.2903245.

[4] *Alice & Bob*, https://www.alice-bob.com/.

[5] B. Apak, M. Bandic, A. Sarkar, and S. Feld, "Ketgpt–dataset augmentation of quantum circuits using transformers", *arXiv preprint arXiv:2402.13352*, 2024.

[6] *Apple unleashes M1*, https://www.apple.com/newsroom/2020/11/apple-unleashes-m1/, 2020.

[7] G. Armeniakos, G. Zervakis, D. Soudris, and J. Henkel, "Hardware approximate techniques for deep neural network accelerators: A survey", *ACM Computing Surveys (CSUR)*, 2022.

[8] K. Arulkumaran, M. P. Deisenroth, M. Brundage, and A. A. Bharath, "Deep reinforcement learning: A brief survey", *IEEE Signal Processing Magazine*, vol. 34, no. 6, pp. 26–38, 2017.

[9] F. Arute, K. Arya, R. Babbush, D. Bacon, J. C. Bardin, R. Barends, R. Biswas, S. Boixo, F. G. Brandao, D. A. Buell, *et al.*, "Quantum supremacy using a programmable superconducting processor", *Nature*, vol. 574, no. 7779, pp. 505–510, 2019.

[10] M. Asghari, A. M. Fathollahi-Fard, S. Mirzapour Al-e-hashem, and M. A. Dulebenets, "Transformation and linearization techniques in optimization: A state-of-the-art survey", *Mathematics*, vol. 10, no. 2, p. 283, 2022.

[11] E. Baek, D. Kwon, and J. Kim, "A multi-neural network acceleration architecture", in *2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA)*, IEEE, 2020, pp. 940–953.

[12] M. Bakator and D. Radosav, "Deep learning and medical diagnosis: A review of literature", *Multimodal Technologies and Interaction*, vol. 2, no. 3, p. 47, 2018.

[13] J. M. Baker, C. Duckering, A. Hoover, and F. T. Chong, "Time-sliced quantum circuit partitioning for modular architectures", in *ACM International Conference on Computing Frontiers*, 2020, pp. 98–107.

[14] I. Baldini *et al.*, "Serverless computing: Current trends and open problems", in *Research Advances in Cloud Computing*. Springer Singapore, 2017, pp. 1–20.

[15] M. Bandic, L. Prielinger, J. Nüßlein, A. Ovide, S. Rodrigo, S. Abadal, H. van Someren, G. Vardoyan, E. Alarcon, C. G. Almudever, *et al.*, "Mapping quantum circuits to modular architectures with qubo", in *IEEE International Conference on Quantum Computing and Engineering (QCE)*, IEEE, vol. 1, 2023, pp. 790–801.

[16] A. Barenco, C. H. Bennett, R. Cleve, D. P. DiVincenzo, N. Margolus, P. Shor, T. Sleator, J. A. Smolin, and H. Weinfurter, "Elementary gates for quantum computation", *Physical review A*, vol. 52, no. 5, p. 3457, 1995.

[17] I. Bello, H. Pham, Q. V. Le, M. Norouzi, and S. Bengio, "Neural combinatorial optimization with reinforcement learning", *arXiv preprint arXiv:1611.09940*, 2016.

[18] K. Bertels, A. Sarkar, T. Hubregtsen, M. Serrao, A. A. Mouedenne, A. Yadav, A. Krol, I. Ashraf, and C. G. Almudever, "Quantum computer architecture toward full-stack quantum accelerators", *IEEE Transactions on Quantum Engineering*, vol. 1, pp. 1–17, 2020.

[19] F. Berto, C. Hua, J. Park, M. Kim, H. Kim, J. Son, H. Kim, J. Kim, and J. Park, "RL4CO: A unified reinforcement learning for combinatorial optimization library", in *NeurIPS 2023 Workshop: New Frontiers in Graph Learning*, https://github.com/ai4co/rl4co, 2023.

[20] J. Biamonte, P. Wittek, N. Pancotti, P. Rebentrost, N. Wiebe, and S. Lloyd, "Quantum machine learning", *Nature*, vol. 549, no. 7671, pp. 195–202, 2017.

[21] N. Binkert, B. Beckmann, G. Black, S. K. Reinhardt, A. Saidi, A. Basu, J. Hestness, D. R. Hower, T. Krishna, S. Sardashti, *et al.*, "The gem5 simulator", *ACM SIGARCH computer architecture news*, vol. 39, no. 2, pp. 1–7, 2011.

[22] F. G. Blanco, E. Russo, M. Palesi, D. Patti, G. Ascia, and V. Catania, "A deep reinforcement learning based online scheduling policy for deep neural network multi-tenant multi-accelerator systems", in *Proceedings of the 61st ACM/IEEE Design Automation Conference*, 2024, pp. 1–6.

[23] S. Boehm, *Lleaves*. [Online]. Available: https://github.com/siboehm/lleaves.

[24] É. Bonnet, T. Miltzow, and P. Rzazewski, "Complexity of token swapping and its variants", *Algorithmica*, vol. 80, pp. 2656–2682, 2018.

[25] A. Botea, A. Kishimoto, and R. Marinescu, "On the complexity of quantum circuit compilation", in *Proceedings of the International Symposium on Combinatorial Search*, vol. 9, 2018, pp. 138–142.

[26] G. Burkard, T. D. Ladd, A. Pan, J. M. Nichol, and J. R. Petta, "Semiconductor spin qubits", *Reviews of Modern Physics*, vol. 95, no. 2, p. 025 003, 2023.

[27] M. Caleffi, M. Amoretti, D. Ferrari, J. Illiano, A. Manzalini, and A. S. Cacciapuoti, "Distributed quantum computing: A survey", *Computer Networks*, vol. 254, p. 110 672, 2024.

[28] A. Callison and N. Chancellor, "Hybrid quantum-classical algorithms in the noisy intermediate-scale quantum era and beyond", *Physical Review A*, vol. 106, no. 1, p. 010 101, 2022.

[29] M. Capra, B. Bussolino, A. Marchisio, G. Masera, M. Martina, and M. Shafique, "Hardware and software optimizations for accelerating deep neural networks: Survey of current trends, challenges, and the road ahead", *IEEE Access*, vol. 8, pp. 225 134–225 180, 2020.

[30] *Cerebras CS-2*, https://cerebras.net/system/, 2021.

[31] M. Cerezo, A. Arrasmith, R. Babbush, S. C. Benjamin, S. Endo, K. Fujii, J. R. McClean, K. Mitarai, X. Yuan, L. Cincio, *et al.*, "Variational quantum algorithms", *Nature Reviews Physics*, vol. 3, no. 9, pp. 625–644, 2021.

[32] R. Chandra, L. Dagum, D. Kohr, R. Menon, D. Maydan, and J. McDonald, *Parallel programming in OpenMP*. Morgan kaufmann, 2001.

[33] J. Chang, J. Wang, B. Li, Y. Zhao, and D. Li, "Attention-based deep reinforcement learning for edge user allocation", *IEEE Transactions on Network and Service Management*, 2023.

[34] E. Charbon, M. Babaie, A. Vladimirescu, and F. Sebastiano, "Cryogenic cmos circuits and systems: Challenges and opportunities in designing the electronic interface for quantum processors", *IEEE Microwave Magazine*, vol. 22, no. 1, pp. 60–78, 2020.

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

[36] Y.-H. Chen, T.-J. Yang, J. Emer, and V. Sze, "Eyeriss v2: A flexible accelerator for emerging deep neural networks on mobile devices", *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, vol. 9, no. 2, pp. 292–308, 2019.

[37] L. Chen, K. Lu, A. Rajeswaran, K. Lee, A. Grover, M. Laskin, P. Abbeel, A. Srinivas, and I. Mordatch, "Decision transformer: Reinforcement learning via sequence modeling", *Advances in neural information processing systems*, vol. 34, pp. 15 084–15 097, 2021.

[38] R. Chen, W. Li, and H. Yang, "A deep reinforcement learning framework based on an attention mechanism and disjunctive graph embedding for the job-shop scheduling problem", *IEEE Transactions on Industrial Informatics*, vol. 19, no. 2, pp. 1322–1331, 2022.

[39] Y. Chen, Y. Xie, L. Song, F. Chen, and T. Tang, "A survey of accelerator architectures for deep neural networks", *Engineering*, vol. 6, no. 3, pp. 264–274, 2020.

[40] Y. Choi and M. Rhu, "Prema: A predictive multi-task scheduling algorithm for preemptible neural processing units", in *2020 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, IEEE, 2020, pp. 220–233.

[41] K. S. Chou, J. Z. Blumoff, C. S. Wang, P. C. Reinhold, C. J. Axline, Y. Y. Gao, L. Frunzio, M. Devoret, L. Jiang, and R. Schoelkopf, "Deterministic teleportation of a quantum gate between two logical qubits", *Nature*, vol. 561, no. 7723, pp. 368–373, 2018.

[42] B. Cottier, R. Rahman, L. Fattorini, N. Maslej, T. Besiroglu, and D. Owen, "The rising costs of training frontier ai models", *arXiv preprint arXiv:2405.21015*, 2024.

[43] D. Cuomo, M. Caleffi, K. Krsulich, F. Tramonto, G. Agliardi, E. Prati, and A. S. Cacciapuoti, "Optimized compiler for distributed quantum computing", *ACM Transactions on Quantum Computing*, vol. 4, no. 2, pp. 1–29, 2023.

[44] L. d'Avossa, M. Caleffi, C. Wang, J. Illiano, S. Zorzetti, and A. S. Cacciapuoti, "Towards the quantum internet: Entanglement rate analysis of high-efficiency electro-optic transducer", in *2023 IEEE International Conference on Quantum Computing and Engineering (QCE)*, IEEE, vol. 1, 2023, pp. 1325–1334.

[45] G. Da Costa, L. Grange, and I. De Courchelle, "Modeling, classifying and generating large-scale google-like workload", *Sustainable Computing: Informatics and Systems*, vol. 19, pp. 305–314, 2018.

[46] A. J. Daley, I. Bloch, C. Kokail, S. Flannigan, N. Pearson, M. Troyer, and P. Zoller, "Practical quantum advantage in quantum simulation", *Nature*, vol. 607, no. 7920, pp. 667–676, 2022.

[47] W. J. Dally, Y. Turakhia, and S. Han, "Domain-specific hardware accelerators", *Communications of the ACM*, vol. 63, no. 7, pp. 48–57, 2020.

[48] A. Das, M. Palesi, J. Kim, and P. P. Pande, "Chip and package-scale interconnects for general-purpose, domain-specific and quantum computing systems-overview, challenges and opportunities", *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, 2024.

[49] A. Das, E. Russo, and M. Palesi, "Multi-objective hardware-mapping co-optimisation for multi-dnn workloads on chiplet-based accelerators", *IEEE Transactions on Computers*, vol. 73, no. 8, pp. 1883–1898, 2024.

[50] S. Dauzère-Pérès, J. Ding, L. Shen, and K. Tamssaouet, "The flexible job shop scheduling problem: A review", *European Journal of Operational Research*, 2023.

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

[52] L. Deng, G. Li, S. Han, L. Shi, and Y. Xie, "Model compression and hardware acceleration for neural networks: A comprehensive survey", *Proceedings of the IEEE*, vol. 108, no. 4, pp. 485–532, 2020.

[53] R. H. Dennard, F. H. Gaensslen, H.-N. Yu, V. L. Rideout, E. Bassous, and A. R. LeBlanc, "Design of ion-implanted mosfet's with very small physical dimensions", *IEEE Journal of solid-state circuits*, vol. 9, no. 5, pp. 256–268, 1974.

[54] E. W. Dijkstra, "A note on two problems in connexion with graphs", in *Edsger Wybe Dijkstra: his life, work, and legacy*, 2022, pp. 287–290.

[55] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, *et al.*, "An image is worth 16x16 words: Transformers for image recognition at scale", *arXiv preprint arXiv:2010.11929*, 2020.

[56] S. Droste, T. Jansen, and I. Wegener, "On the analysis of the (1+ 1) evolutionary algorithm", *Theoretical Computer Science*, vol. 276, no. 1-2, pp. 51–81, 2002.

[57] Z. Du, R. Fasthuber, T. Chen, P. Ienne, L. Li, T. Luo, X. Feng, Y. Chen, and O. Temam, "Shidiannao: Shifting vision processing closer to the sensor", in *Proceedings of the 42nd Annual International Symposium on Computer Architecture*, 2015, pp. 92–104.

[58] V. P. Dwivedi, C. K. Joshi, A. T. Luu, T. Laurent, Y. Bengio, and X. Bresson, "Benchmarking graph neural networks", *Journal of Machine Learning Research*, vol. 24, no. 43, pp. 1–48, 2023.

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

[60] M. Emani, Z. Xie, S. Raskar, V. Sastry, W. Arnold, B. Wilson, R. Thakur, V. Vishwanath, Z. Liu, M. E. Papka, *et al.*, "A comprehensive evaluation of novel ai accelerators for deep learning workloads", in *2022 IEEE/ACM international workshop on performance modeling, benchmarking and simulation of high performance computer systems (PMBS)*, IEEE, 2022, pp. 13–25.

[61] P. Escofet, A. Ovide, C. G. Almudever, E. Alarcón, and S. Abadal, "Hungarian qubit assignment for optimized mapping of quantum circuits on multi-core architectures", *IEEE Computer Architecture Letters*, vol. 22, no. 2, pp. 161–164, 2023.

[62] P. Escofet, A. Ovide, M. Bandic, L. Prielinger, H. van Someren, S. Feld, E. Alarcón, S. Abadal, and C. G. Almudéver, "Revisiting the mapping of quantum circuits: Entering the multi-core era", *ACM Transactions on Quantum Computing*, 2024.

[63] P. Escofet, S. B. Rached, S. Rodrigo, C. G. Almudever, E. Alarcón, and S. Abadal, "Interconnect fabrics for multi-core quantum processors: A context analysis", in *Proceedings of the 16th International Workshop on Network on Chip Architectures*, 2023, pp. 34–39.

[64] H. Esmaeilzadeh, E. Blem, R. St. Amant, K. Sankaralingam, and D. Burger, "Dark silicon and the end of multicore scaling", in *Proceedings of the 38th annual international symposium on Computer architecture*, 2011, pp. 365–376.

[65] N. Fasfous, M. R. Vemparala, A. Frickenstein, E. Valpreda, D. Salihu, J. Höfer, A. Singh, N.-S. Nagaraja, H.-J. Voegel, N. A. V. Doan, *et al.*, "Anaconga: Analytical hw-cnn co-design using nested genetic algorithms", in *2022 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, IEEE, 2022, pp. 238–243.

[66] D. Ferrari, A. S. Cacciapuoti, M. Amoretti, and M. Caleffi, "Compiler design for distributed quantum computing", *IEEE Transactions on Quantum Engineering*, vol. 2, pp. 1–20, 2021.

[67] D. Ferrari, S. Carretta, and M. Amoretti, "A modular quantum compilation framework for distributed quantum computing", *IEEE Transactions on Quantum Engineering*, vol. 4, pp. 1–13, 2023.

[68] M. Fey and J. E. Lenssen, "Fast graph representation learning with PyTorch Geometric", in *ICLR Workshop on Representation Learning on Graphs and Manifolds*, 2019.

[69] R. Feynman, "There's plenty of room at the bottom", in *Feynman and computation*, CRC Press, 2018, pp. 63–76.

[70] R. Garg, E. Qin, F. Muñoz-Matrínez, R. Guirado, A. Jain, S. Abadal, J. L. Abellán, M. E. Acacio, E. Alarcón, S. Rajamanickam, *et al.*, "Understanding the design-space of sparse/dense multiphase gnn dataflows on spatial accelerators", in *2022 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, IEEE, 2022, pp. 571–582.

[71] M. Gen, R. Cheng, and D. Wang, "Genetic algorithms for solving shortest path problems", in *IEEE International Conference on Evolutionary Computation*, IEEE, 1997, pp. 401–406.

[72] T. Geng, A. Li, R. Shi, C. Wu, T. Wang, Y. Li, P. Haghi, A. Tumeo, S. Che, S. Reinhardt, *et al.*, "Awb-gcn: A graph convolutional network accelerator with runtime workload rebalancing", in *2020 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, IEEE, 2020, pp. 922–936.

[73] S. Ghodrati, B. H. Ahn, J. K. Kim, S. Kinzer, B. R. Yatham, N. Alla, H. Sharma, M. Alian, E. Ebrahimi, N. S. Kim, *et al.*, "Planaria: Dynamic architecture fission for spatial multi-tenant acceleration of deep neural networks", in *2020 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, IEEE, 2020, pp. 681–697.

[74] L. Giannelli, P. Sgroi, J. Brown, G. S. Paraoanu, M. Paternostro, E. Paladino, and G. Falci, "A tutorial on optimal control and reinforcement learning methods for quantum technologies", *Physics Letters A*, vol. 434, p. 128 054, 2022.

[75] F. Glover and E. Woolsey, "Further reduction of zero-one polynomial programming problems to zero-one linear programming problems", *Operations Research*, vol. 21, no. 1, pp. 156–161, 1973.

[76] Google, *Quantum AI Roadmap*, https://quantumai.google/roadmap, Accessed in August 2025.

[77] *Google Cloud TPUv4*, https://cloud.google.com/tpu/, 2021.

[78] *Google Edge TPUv1*, https://cloud.google.com/edge-tpu/, 2018.

[79] *Google Quantum AI*, https://quantumai.google/.

[80] Google Quantum AI, "Suppressing quantum errors by scaling a surface code logical qubit", *Nature*, vol. 614, no. 7949, pp. 676–681, 2023.

[81] R. L. Graham, E. L. Lawler, J. K. Lenstra, and A. R. Kan, "Optimization and approximation in deterministic sequencing and scheduling: A survey", in *Annals of discrete mathematics*, vol. 5, Elsevier, 1979, pp. 287–326.

[82] S. Grigorescu, B. Trasnea, T. Cocias, and G. Macesanu, "A survey of deep learning techniques for autonomous driving", *Journal of Field Robotics*, vol. 37, no. 3, pp. 362–386, 2020.

[83] L. K. Grover, "A fast quantum mechanical algorithm for database search", in *ACM symposium on Theory of computing*, 1996, pp. 212–219.

[84] Gurobi Optimization, LLC, *Gurobi Optimizer Reference Manual*, 2023. [Online]. Available: https://www.gurobi.com.

[85] K. E. Hamilton, C. D. Schuman, S. R. Young, R. S. Bennink, N. Imam, and T. S. Humble, "Accelerating scientific computing in the post-moore's era", *ACM Transactions on Parallel Computing (TOPC)*, vol. 7, no. 1, pp. 1–31, 2020.

[86] W. Hamilton, Z. Ying, and J. Leskovec, "Inductive representation learning on large graphs", *Advances in neural information processing systems*, vol. 30, 2017.

[87] G. R. Harik, F. G. Lobo, and D. E. Goldberg, "The compact genetic algorithm", *IEEE transactions on evolutionary computation*, vol. 3, no. 4, pp. 287–297, 1999.

[88] P. Hauke, H. G. Katzgraber, W. Lechner, H. Nishimori, and W. D. Oliver, "Perspectives of quantum annealing: Methods and implementations", *Reports on Progress in Physics*, vol. 83, no. 5, p. 054 401, 2020.

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

[90] J. L. Hennessy and D. A. Patterson, *Computer architecture: a quantitative approach*. Elsevier, 2011.

[91] J. L. Hennessy and D. A. Patterson, "A new golden age for computer architecture", *Communications of the ACM*, vol. 62, no. 2, pp. 48–60, 2019.

[92] L. Henriet, L. Beguin, A. Signoles, T. Lahaye, A. Browaeys, G.-O. Reymond, and C. Jurczak, "Quantum computing with neutral atoms", *Quantum*, vol. 4, p. 327, 2020.

[93] C. Heunen and P. A. Martinez, "Automated distribution of quantum circuits", *Physical Review A*, vol. 100, p. 032 308, 2019.

[94] S. Hillmich, A. Zulehner, and R. Wille, "Exploiting quantum teleportation in quantum circuit mapping", in *Proceedings of the 26th Asia and South Pacific Design Automation Conference*, 2021, pp. 792–797.

[95] P. Hopf, N. Quetschlich, L. Schulz, and R. Wille, "Improving figures of merit for quantum circuit compilation", in *2025 Design, Automation & Test in Europe Conference (DATE)*, 2025, pp. 1–7.

[96] M. Horeni, P. Taheri, P.-A. Tsai, A. Parashar, J. Emer, and S. Joshi, "Ruby: Improving hardware efficiency for tensor algebra accelerators through imperfect factorization", in *2022 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, IEEE, 2022, pp. 254–266.

[97] M. Horowitz, "1.1 computing's energy problem (and what we can do about it)", in *2014 IEEE International Solid-State Circuits Conference Digest of Technical Papers (ISSCC)*, IEEE, 2014, pp. 10–14.

[98] C.-Y. Huang, C.-H. Lien, and W.-K. Mak, "Reinforcement learning and dear framework for solving the qubit mapping problem", in *41st IEEE/ACM international conference on computer-aided design*, 2022, pp. 1–9.

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

[100] IBM, *Quantum Roadmap*, https://www.ibm.com/roadmaps/quantum/, Accessed in August 2025.

[101] *IBM Quantum*, https://www.ibm.com/quantum.

[102] E. M. Ibrahim, L. Mei, and M. Verhelst, "Taxonomy and benchmarking of precision-scalable mac arrays under enhanced dnn dataflow representation", *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 69, no. 5, pp. 2013–2024, 2022.

[103] IEA, *Energy and AI*, https://www.iea.org/reports/energy-and-ai, CC BY 4.0, 2025.

[104] A. Javadi-Abhari, M. Treinish, K. Krsulich, C. J. Wood, J. Lishman, J. Gacon, S. Martiel, P. D. Nation, L. S. Bishop, A. W. Cross, B. R. Johnson, and J. M. Gambetta, *Quantum computing with Qiskit*, 2024. DOI: 10.48550/arXiv.2405.08810. arXiv: 2405.08810 [quant-ph].

[105] H. Jnane, B. Undseth, Z. Cai, S. C. Benjamin, and B. Koczor, "Multicore quantum computing", *Physical Review Applied*, vol. 18, no. 4, 2022.

[106] E. Joos and H. D. Zeh, "The emergence of classical properties through interaction with the environment", *Zeitschrift für Physik B Condensed Matter*, vol. 59, no. 2, pp. 223–243, 1985.

[107] N. P. Jouppi, D. H. Yoon, M. Ashcraft, M. Gottscho, T. B. Jablin, G. Kurian, J. Laudon, S. Li, P. Ma, X. Ma, *et al.*, "Ten lessons from three generations shaped google's tpuv4i: Industrial product", in *2021 ACM/IEEE 48th Annual International Symposium on Computer Architecture (ISCA)*, IEEE, 2021, pp. 1–14.

[108] V. J. Jung, A. Symons, L. Mei, M. Verhelst, and L. Benini, "Salsa: Simulated annealing based loop-ordering scheduler for dnn accelerators", in *2023 IEEE 5th International Conference on Artificial Intelligence Circuits and Systems (AICAS)*, IEEE, 2023, pp. 1–5.

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

[110] S.-C. Kao and T. Krishna, "Magma: An optimization framework for mapping multiple dnns on multiple accelerator cores", in *2022 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, IEEE, 2022, pp. 814–830.

[111] S.-C. Kao, A. Parashar, P.-A. Tsai, and T. Krishna, "Demystifying map space exploration for npus", in *2022 IEEE International Symposium on Workload Characterization (IISWC)*, IEEE, 2022, pp. 269–281.

[112] S.-C. Kao, M. Pellauer, A. Parashar, and T. Krishna, "Digamma: Domain-aware genetic algorithm for hw-mapping co-optimization for dnn accelerators", in *2022 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, IEEE, 2022, pp. 232–237.

[113] J. Kaplan, S. McCandlish, T. Henighan, T. B. Brown, B. Chess, R. Child, S. Gray, A. Radford, J. Wu, and D. Amodei, "Scaling laws for neural language models", *arXiv preprint arXiv:2001.08361*, 2020.

[114] P. K. Kashyap, S. Kumar, A. Jaiswal, O. Kaiwartya, M. Kumar, U. Dohare, and A. H. Gandomi, "Decent: Deep learning enabled green computation for edge centric 6g networks", *IEEE Transactions on Network and Service Management*, pp. 1–1, 2022. DOI: 10.1109/TNSM.2022.3145056.

[115] G. Ke, Q. Meng, T. Finley, T. Wang, W. Chen, W. Ma, Q. Ye, and T.-Y. Liu, "Lightgbm: A highly efficient gradient boosting decision tree", *Advances in neural information processing systems*, vol. 30, pp. 3146–3154, 2017.

[116] A. B. Keha, I. R. de Farias Jr, and G. L. Nemhauser, "Models for representing piecewise linear cost functions", *Operations Research Letters*, vol. 32, no. 1, pp. 44–48, 2004.

[117] H. Kim, M. Kim, F. Berto, J. Kim, and J. Park, "Devformer: A symmetric transformer for context-aware device placement", in *International Conference on Machine Learning*, PMLR, 2023, pp. 16 541–16 566.

[118] S. Kim, H. Genc, V. V. Nikiforov, K. Asanović, B. Nikolić, and Y. S. Shao, "Moca: Memory-centric, adaptive execution for multi-tenant deep neural networks", in *2023 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, IEEE, 2023, pp. 828–841.

[119] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks", *CoRR*, vol. abs/1609.02907, 2016. arXiv: 1609.02907. [Online]. Available: http://arxiv.org/abs/1609.02907.

[120] A. Y. Kitaev, "Quantum computations: Algorithms and error correction", *Russian Mathematical Surveys*, vol. 52, no. 6, p. 1191, 1997.

[121] A. Y. Kitaev, "Fault-tolerant quantum computation by anyons", *Annals of physics*, vol. 303, no. 1, pp. 2–30, 2003.

[122] J. Koch, T. M. Yu, J. Gambetta, A. A. Houck, D. I. Schuster, J. Majer, A. Blais, M. H. Devoret, S. M. Girvin, and R. J. Schoelkopf, "Charge-insensitive qubit design derived from the cooper pair box", *Physical Review A—Atomic, Molecular, and Optical Physics*, vol. 76, no. 4, p. 042 319, 2007.

[123] S. P. Kolodziej, M. Aznaveh, M. Bullock, J. David, T. A. Davis, M. Henderson, Y. Hu, and R. Sandstrom, "The suitesparse matrix collection website interface", *Journal of Open Source Software*, vol. 4, no. 35, p. 1244, 2019. DOI: 10.21105/joss.01244. [Online]. Available: https://doi.org/10.21105/joss.01244.

[124] W. Kool, H. Van Hoof, and M. Welling, "Attention, learn to solve routing problems!", *arXiv preprint arXiv:1803.08475*, 2018.

[125] D. Kremer, V. Villar, H. Paik, I. Duran, I. Faro, and J. Cruz-Benito, "Practical and efficient quantum circuit synthesis and transpiling with reinforcement learning", *arXiv preprint arXiv:2405.13196*, 2024.

[126] H. Kwon, P. Chatarasi, M. Pellauer, A. Parashar, V. Sarkar, and T. Krishna, "Understanding reuse, performance, and hardware cost of dnn dataflow: A data-centric approach", in *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture*, 2019, pp. 754–768.

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

[128] H. Kwon, L. Lai, M. Pellauer, T. Krishna, Y.-H. Chen, and V. Chandra, "Heterogeneous dataflow accelerators for multi-dnn workloads", in *2021 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, IEEE, 2021, pp. 71–83.

[129] H. Kwon, A. Samajdar, and T. Krishna, "Maeri: Enabling flexible dataflow mapping over dnn accelerators via reconfigurable interconnects", *ACM SIGPLAN Notices*, vol. 53, no. 2, pp. 461–475, 2018.

[130] S. Kwon, A. Tomonaga, G. Lakshmi Bhai, S. J. Devitt, and J.-S. Tsai, "Gate-based superconducting quantum computing", *Journal of Applied Physics*, vol. 129, no. 4, 2021.

[131] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning", *nature*, vol. 521, no. 7553, pp. 436–444, 2015.

[132] C. E. Leiserson, N. C. Thompson, J. S. Emer, B. C. Kuszmaul, B. W. Lampson, D. Sanchez, and T. B. Schardl, "There's plenty of room at the top: What will drive computer performance after moore's law?", *Science*, vol. 368, no. 6495, 2020.

[133] J. K. Lenstra, A. R. Kan, and P. Brucker, "Complexity of machine scheduling problems", in *Annals of discrete mathematics*, vol. 1, Elsevier, 1977, pp. 343–362.

[134] S. Levine, P. Pastor, A. Krizhevsky, J. Ibarz, and D. Quillen, "Learning hand-eye coordination for robotic grasping with deep learning and large-scale data collection", *The International journal of robotics research*, vol. 37, no. 4-5, pp. 421–436, 2018.

[135] G. Li, Y. Ding, and Y. Xie, "Tackling the qubit mapping problem for nisq-era quantum devices", in *Proceedings of the twenty-fourth international conference on architectural support for programming languages and operating systems*, 2019, pp. 1001–1014.

[136] W. Li, H. Luo, Z. Lin, C. Zhang, Z. Lu, and D. Ye, "A survey on transformers in reinforcement learning", *arXiv preprint arXiv:2301.03044*, 2023.

[137] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning", *arXiv:1509.02971*, 2015.

[138] W.-H. Lin, J. Kimko, B. Tan, N. Bjørner, and J. Cong, "Scalable optimal layout synthesis for nisq quantum processors", in *ACM/IEEE Design Automation Conference (DAC)*, IEEE, 2023, pp. 1–6.

[139] A. Lindermayr, N. Megow, and M. Rapp, "Speed-oblivious online scheduling: Knowing (precise) speeds is not necessary", in *International Conference on Machine Learning*, PMLR, 2023, pp. 21 312–21 334.

[140] Z. Liu, J. Leng, Z. Zhang, Q. Chen, C. Li, and M. Guo, "Veltair: Towards high-performance multi-tenant deep learning services via adaptive compilation and scheduling", in *Proceedings of the 27th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*, 2022, pp. 388–401.

[141] W. Lu, G. Yan, J. Li, S. Gong, Y. Han, and X. Li, "Flexflow: A flexible dataflow accelerator architecture for convolutional neural networks", in *2017 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, IEEE, 2017, pp. 553–564.

[142] D. Main, P. Drmota, D. Nadlinger, E. Ainley, A. Agrawal, B. Nichol, R. Srinivas, G. Araneda, and D. Lucas, "Distributed quantum computing across an optical network link", *Nature*, pp. 1–6, 2025.

[143] J. R. McClean, J. Romero, R. Babbush, and A. Aspuru-Guzik, "The theory of variational hybrid quantum-classical algorithms", *New Journal of Physics*, vol. 18, no. 2, p. 023 023, 2016.

[144] L. Mei, P. Houshmand, V. Jain, S. Giraldo, and M. Verhelst, "Zigzag: Enlarging joint architecture-mapping design space exploration for dnn accelerators", *IEEE Transactions on Computers*, vol. 70, no. 8, pp. 1160–1174, 2021.

[145] L. Mei, H. Liu, T. Wu, H. E. Sumbul, M. Verhelst, and E. Beigne, "A uniform latency model for dnn accelerators with diverse architectures and dataflows", in *2022 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, IEEE, 2022, pp. 220–225.

[146] *Metaverse*, https://about.meta.com/metaverse, Accessed: 2025-08-01.

[147] A. Montanaro, "Quantum algorithms: An overview", *npj Quantum Information*, vol. 2, no. 1, pp. 1–8, 2016.

[148] G. E. Moore *et al.*, *Cramming more components onto integrated circuits*, 1965.

[149] C. Morris, N. M. Kriege, F. Bause, K. Kersting, P. Mutzel, and M. Neumann, "Tudataset: A collection of benchmark datasets for learning with graphs", in *ICML 2020 Workshop on Graph Representation Learning and Beyond (GRL+ 2020)*, 2020. arXiv: 2007.08663. [Online]. Available: www.graphlearning.io.

[150] F. Muñoz-Martínez, J. L. Abellán, M. E. Acacio, and T. Krishna, "Stonne: Enabling cycle-level microarchitectural simulation for dnn inference accelerators", in *2021 IEEE International Symposium on Workload Characterization (IISWC)*, IEEE, 2021, pp. 201–213.

[151] P. Murali, D. M. Debroy, K. R. Brown, and M. Martonosi, "Architecting noisy intermediate-scale trapped ion quantum computers", in *2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA)*, IEEE, 2020, pp. 529–542.

[152] G. Nannicini, L. S. Bishop, O. Günlük, and P. Jurcevic, "Optimal qubit assignment and routing via integer programming", *ACM Transactions on Quantum Computing*, vol. 4, no. 1, pp. 1–31, 2022.

[153] M. Naumov, D. Mudigere, H.-J. M. Shi, J. Huang, N. Sundaraman, J. Park, X. Wang, U. Gupta, C.-J. Wu, A. G. Azzolini, *et al.*, "Deep learning recommendation model for personalization and recommendation systems", *arXiv preprint arXiv:1906.00091*, 2019.

[154] C. Nayak, S. H. Simon, A. Stern, M. Freedman, and S. Das Sarma, "Non-abelian anyons and topological quantum computation", *Reviews of Modern Physics*, vol. 80, no. 3, pp. 1083–1159, 2008.

[155] M. A. Nielsen and I. L. Chuang, *Quantum computation and quantum information*. Cambridge university press, 2010.

[156] R. J. Nowling and H. Mauch, "Priority encoding scheme for solving permutation and constraint problems with genetic algorithms and simulated annealing", in *International Conference on Information Technology: New Generations*, IEEE, 2011, pp. 810–815.

[157] *NVDLA: The NVIDIA Deep Learning Accelerator*, http://nvdla.org/.

[158] Y. H. Oh, Y. Jin, T. J. Ham, and J. W. Lee, "Layerweaver+: A qos-aware layer-wise dnn scheduler for multi-tenant neural processing units", *IEICE TRANSACTIONS on Information and Systems*, vol. 105, no. 2, pp. 427–431, 2022.

[159] Y. H. Oh, S. Kim, Y. Jin, S. Son, J. Bae, J. Lee, Y. Park, D. U. Kim, T. J. Ham, and J. W. Lee, "Layerweaver: Maximizing resource utilization of neural processing units via layer-wise scheduling", in *2021 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, 2021, pp. 584–597.

[160] *Open Neural Network Exchange (ONNX)*, https://github.com/onnx/onnx, 2017.

[161] E. Osaba and E. Villar, *Qoptlib: A quantum computing oriented benchmark for combinatorial optimization problems*, version V1, 2023. DOI: 10.17632/h32z9kcz3s.1.

[162] G. Padda, E. Tham, A. Brodutch, and D. Touchette, "Improving qubit routing by using entanglement mediated remote gates", in *2024 IEEE International Conference on Quantum Computing and Engineering (QCE)*, IEEE, vol. 1, 2024, pp. 1770–1776.

[163] M. Pandey, M. Fernandez, F. Gentile, O. Isayev, A. Tropsha, A. C. Stern, and A. Cherkasov, "The transformational role of gpu computing and deep learning in drug discovery", *Nature Machine Intelligence*, vol. 4, no. 3, pp. 211–221, 2022.

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

[165] A. Parashar, M. Rhu, A. Mukkara, A. Puglielli, R. Venkatesan, B. Khailany, J. Emer, S. W. Keckler, and W. J. Dally, "Scnn: An accelerator for compressed-sparse convolutional neural networks", *ACM SIGARCH computer architecture news*, vol. 45, no. 2, pp. 27–40, 2017.

[166] T. Park and C. Y. Lee, "Algorithms for partitioning a graph", *Computers & Industrial Engineering*, vol. 28, no. 4, pp. 899–909, 1995.

[167] A. Parra-Rodriguez, P. Lougovski, L. Lamata, E. Solano, and M. Sanz, "Digital-analog quantum computation", *Physical Review A*, vol. 101, no. 2, p. 022 305, 2020.

[168] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, *et al.*, "Pytorch: An imperative style, high-performance deep learning library", *Advances in neural information processing systems*, vol. 32, 2019.

[169] D. Patterson, J. Gonzalez, Q. Le, C. Liang, L.-M. Munguia, D. Rothchild, D. So, M. Texier, and J. Dean, "Carbon emissions and large neural network training", *arXiv preprint arXiv:2104.10350*, 2021.

[170] L. Perron and V. Furnon, *Or-tools*, version v9.10, Google, May 7, 2024. [Online]. Available: https://developers.google.com/optimization/.

[171] C. Phillips, C. Stein, and J. Wein, "Minimizing average completion time in the presence of release dates", *Mathematical Programming*, vol. 82, no. 1, pp. 199–223, 1998.

[172] J. Preskill, "Quantum computing in the nisq era and beyond", *Quantum*, vol. 2, p. 79, 2018.

[173] P. Promponas, A. Mudvari, L. Della Chiesa, P. Polakos, L. Samuel, and L. Tassiulas, "Compiler for distributed quantum computing: A reinforcement learning approach", *arXiv preprint arXiv:2404.17077*, 2024.

[174] PsiQuantum, "A manufacturable platform for photonic quantum computing", *Nature*, vol. 641, no. 8064, pp. 876–883, 2025.

[175] P. Puigdemont, E. Russo, A. Wassington, A. Das, S. Abadal, and M. Palesi, "A data-driven approach to dataflow-aware online scheduling for graph neural network inference", in *Proceedings of the 30th Asia and South Pacific Design Automation Conference*, 2025, pp. 1195–1201.

[176] E. Qin, A. Samajdar, H. Kwon, V. Nadella, S. Srinivasan, D. Das, B. Kaul, and T. Krishna, "Sigma: A sparse and irregular gemm accelerator with flexible interconnects for dnn training", in *2020 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, IEEE, 2020, pp. 58–70.

[177] Y. Qin, S. Hu, Y. Lin, W. Chen, N. Ding, G. Cui, Z. Zeng, X. Zhou, Y. Huang, C. Xiao, *et al.*, "Tool learning with foundation models", *ACM Computing Surveys*, vol. 57, no. 4, pp. 1–40, 2024.

[178] Qiskit contributors, *Qiskit: An open-source framework for quantum computing*, 2023. DOI: 10.5281/zenodo.2573505.

[179] N. Quetschlich, L. Burgholzer, and R. Wille, "Mqt bench: Benchmarking software and design automation tools for quantum computing", *Quantum*, vol. 7, p. 1062, 2023.

[180] V. J. Reddi, C. Cheng, D. Kanter, P. Mattson, G. Schmuelling, C.-J. Wu, B. Anderson, M. Breughe, M. Charlebois, W. Chou, *et al.*, "MLPerf Inference Benchmark", in *Proceedings of the Annual ACM/IEEE International Symposium on Computer Architecture*, 2020, pp. 446–459.

[181] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection", in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 779–788.

[182] S. Resch and U. R. Karpuzcu, "Benchmarking quantum computers and the impact of quantum noise", *ACM Computing Surveys (CSUR)*, vol. 54, no. 7, pp. 1–35, 2021.

[183] *Rigetti*, https://www.rigetti.com/.

[184] S. Rodrigo, S. Abadal, C. G. Almudéver, and E. Alarcón, "Modelling short-range quantum teleportation for scalable multi-core quantum computing architectures", in *ACM International Conference on Nanoscale Computing and Communication*, 2021, pp. 1–7.

[185] F. Romero, Q. Li, N. J. Yadwadkar, and C. Kozyrakis, "INFaaS: Automated model-less inference serving", in *2021 USENIX Annual Technical Conference (USENIX ATC 21)*, 2021, pp. 397–411.

[186] O. Roudenko and M. Schoenauer, "A steady performance stopping criterion for pareto-based evolutionary algorithms", in *6th International Multi-Objective Programming and Goal Programming Conference*, 2004.

[187] E. Russo, F. G. Blanco, M. Palesi, G. Ascia, D. Patti, and V. Catania, "Towards fair and firm real-time scheduling in dnn multi-tenant multi-accelerator systems via reinforcement learning", in *2024 IEEE International Symposium on Circuits and Systems (ISCAS)*, IEEE, 2024, pp. 1–5.

[188] E. Russo, M. Palesi, G. Ascia, D. Patti, S. Monteleone, and V. Catania, "Memory-aware dnn algorithm-hardware mapping via integer linear programming", in *Proceedings of the 20th ACM International Conference on Computing Frontiers*, 2023, pp. 134–143.

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

[190] E. Russo, M. Palesi, S. Monteleone, D. Patti, G. Ascia, and V. Catania, "Medea: A multi-objective evolutionary approach to dnn hardware mapping", in *2022 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, IEEE, 2022, pp. 226–231.

[191] E. Russo, M. Palesi, S. Monteleone, D. Patti, H. Lahdhiri, G. Ascia, and V. Catania, "Exploiting the approximate computing paradigm with dnn hardware accelerators", in *2022 11th Mediterranean Conference on Embedded Computing (MECO)*, IEEE, 2022, pp. 1–4.

[192] E. Russo, M. Palesi, S. Monteleone, D. Patti, A. Mineo, G. Ascia, and V. Catania, "Dnn model compression for iot domain-specific hardware accelerators", *IEEE Internet of Things Journal*, vol. 9, no. 9, pp. 6650–6662, 2021.

[193] E. Russo, M. Palesi, D. Patti, G. Ascia, and V. Catania, "Optimizing qubit assignment in modular quantum systems via attention-based deep reinforcement learning", in *2025 Design, Automation & Test in Europe Conference (DATE)*, IEEE, 2025, pp. 1–7.

[194] E. Russo, M. Palesi, D. Patti, S. Monteleone, G. Ascia, and V. Catania, "Multi-objective end-to-end design space exploration of parameterized dnn accelerators", *IEEE Internet of Things Journal*, pp. 1–1, 2022. DOI: 10.1109/JIOT.2022.3209401.

[195] R. Sarkar, S. Abi-Karam, Y. He, L. Sathidevi, and C. Hao, "Flowgnn: A dataflow architecture for real-time workload-agnostic graph neural network inference", in *2023 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, IEEE, 2023, pp. 1099–1112.

[196] M. Sarovar, T. Proctor, K. Rudinger, K. Young, E. Nielsen, and R. Blume-Kohout, "Detecting crosstalk errors in quantum information processors", *Quantum*, vol. 4, p. 321, 2020.

[197] L. Schrage, "Letter to the editor—a proof of the optimality of the shortest remaining processing time discipline", *Operations Research*, vol. 16, no. 3, pp. 687–690, 1968. DOI: 10.1287/opre.16.3.687. [Online]. Available: https://doi.org/10.1287/opre.16.3.687.

[198] A. S. Schulz and M. Skutella, "Scheduling unrelated machines by randomized rounding", *SIAM Journal on Discrete Mathematics*, vol. 15, no. 4, pp. 450–469, 2002.

[199] C. D. Schuman, S. R. Kulkarni, M. Parsa, J. P. Mitchell, B. Kay, *et al.*, "Opportunities for neuromorphic computing algorithms and applications", *Nature Computational Science*, vol. 2, no. 1, pp. 10–19, 2022.

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

[201] P. Shaw, J. Uszkoreit, and A. Vaswani, "Self-attention with relative position representations", *arXiv preprint arXiv:1803.02155*, 2018.

[202] P. W. Shor, "Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer", *SIAM review*, vol. 41, no. 2, pp. 303–332, 1999.

[203] M. Y. Siraichi, V. F. d. Santos, C. Collange, and F. M. Q. Pereira, "Qubit allocation", in *International Symposium on Code Generation and Optimization*, 2018, pp. 113–125.

[204] K. N. Smith, G. S. Ravi, J. M. Baker, and F. T. Chong, "Scaling superconducting quantum computers with chiplet architectures", in *IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2022, pp. 1092–1109.

[205] J. Soifer, J. Li, M. Li, J. Zhu, Y. Li, Y. He, E. Zheng, A. Oltean, M. Mosyak, C. Barnes, *et al.*, "Deep learning inference service at microsoft", in *Conference on Operational Machine Learning*, 2019, pp. 15–17.

[206] O. Spantidi and I. Anagnostopoulos, "How much is too much error? analyzing the impact of approximate multipliers on dnns", in *2022 23rd International Symposium on Quality Electronic Design (ISQED)*, IEEE, 2022, pp. 1–6.

[207] M. Steffen, D. P. DiVincenzo, J. M. Chow, T. N. Theis, and M. B. Ketchen, "Quantum computing: An ibm perspective", *IBM Journal of Research and Development*, vol. 55, no. 5, pp. 13–1, 2011.

[208] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press, 2018.

[209] A. Symons, L. Mei, S. Colleman, P. Houshmand, S. Karl, and M. Verhelst, "Towards heterogeneous multi-core accelerators exploiting fine-grained scheduling of layer-fused deep neural networks", *arXiv:2212.10612*, 2022.

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

[211] V. Sze, Y.-H. Chen, T.-J. Yang, and J. S. Emer, "Efficient processing of deep neural networks", *Synthesis Lectures on Computer Architecture*, vol. 15, no. 2, pp. 1–341, 2020.

[212] V. Sze, Y.-H. Chen, T.-J. Yang, and J. S. Emer, "How to evaluate deep neural network processors: Tops/w (alone) considered harmful", *IEEE Solid-State Circuits Magazine*, vol. 12, no. 3, pp. 28–41, 2020.

[213] *Tesla Autopilot*, https://www.tesla.com/autopilot, Accessed: 2025-08-01.

[214] J. Torrejon, M. Riou, F. A. Araujo, S. Tsunegi, G. Khalsa, D. Querlioz, P. Bortolotti, V. Cros, K. Yakushiji, A. Fukushima, *et al.*, "Neuromorphic computing with nanoscale spintronic oscillators", *Nature*, vol. 547, no. 7664, pp. 428–431, 2017.

[215] E. Valpreda, P. Morı, N. Fasfous, M. R. Vemparala, A. Frickenstein, L. Frickenstein, W. Stechele, C. Passerone, G. Masera, and M. Martina, "Hw-flow-fusion: Inter-layer scheduling for convolutional neural network accelerators with dataflow architectures", *Electronics*, vol. 11, no. 18, p. 2933, 2022.

[216] A. Van Den Oord, S. Dieleman, H. Zen, K. Simonyan, O. Vinyals, A. Graves, N. Kalchbrenner, A. Senior, K. Kavukcuoglu, *et al.*, "Wavenet: A generative model for raw audio", *arXiv preprint arXiv:1609.03499*, vol. 12, p. 1, 2016.

[217] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need", *Advances in neural information processing systems*, vol. 30, 2017.

[218] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Liò, and Y. Bengio, "Graph attention networks", in *International Conference on Learning Representations*, 2018. [Online]. Available: https://openreview.net/forum?id=rJXMpikCZ.

[219] M. Verhelst, L. Benini, and N. Verma, "How to keep pushing ml accelerator performance? know your rooflines!", *IEEE Journal of Solid-State Circuits*, 2025.

[220] O. Vinyals, S. Bengio, and M. Kudlur, "Order matters: Sequence to sequence for sets", *arXiv preprint arXiv:1511.06391*, 2015.

[221] O. Vinyals, M. Fortunato, and N. Jaitly, "Pointer networks", *Advances in neural information processing systems*, vol. 28, 2015.

[222] M. M. Waldrop, "More than moore", *Nature*, vol. 530, no. 7589, pp. 144–148, 2016.

[223] H.-n. Wang, N. Liu, Y.-y. Zhang, D.-w. Feng, F. Huang, D.-s. Li, and Y.-m. Zhang, "Deep reinforcement learning: A survey", *Frontiers of Information Technology & Electronic Engineering*, vol. 21, no. 12, pp. 1726–1744, 2020.

[224] A. Wassington and S. Abadal, "Bias reduction via cooperative bargaining in synthetic graph dataset generation", *Applied Intelligence*, vol. 55, no. 2, p. 130, 2025.

[225] R. Wille, L. Burgholzer, and A. Zulehner, "Mapping quantum circuits to ibm qx architectures using the minimal number of swap and h operations", in *Proceedings of the 56th Annual Design Automation Conference 2019*, 2019, pp. 1–6.

[226] R. J. Williams, "Simple statistical gradient-following algorithms for connectionist reinforcement learning", *Machine learning*, vol. 8, pp. 229–256, 1992.

[227] T. Wolf, L. Debut, V. Sanh, J. Chaumond, C. Delangue, A. Moi, P. Cistac, T. Rault, R. Louf, M. Funtowicz, *et al.*, "Transformers: State-of-the-art natural language processing", in *Proceedings of the 2020 conference on empirical methods in natural language processing: system demonstrations*, 2020, pp. 38–45.

[228] W. K. Wootters and W. H. Zurek, "The no-cloning theorem", *Physics Today*, vol. 62, no. 2, pp. 76–77, 2009.

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

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

[231] X. Xu, Y. Ding, S. X. Hu, M. Niemier, J. Cong, Y. Hu, and Y. Shi, "Scaling for edge inference of deep neural networks", *Nature Electronics*, vol. 1, no. 4, pp. 216–222, 2018.

[232] Y. Xu, K. Li, J. Hu, and K. Li, "A Genetic Algorithm for Task Scheduling on Heterogeneous Computing Systems using Multiple Priority Queues", *Elsevier Information Sciences*, vol. 270, pp. 255–287, 2014.

[233] M. Yan, L. Deng, X. Hu, L. Liang, Y. Feng, X. Ye, Z. Zhang, D. Fan, and Y. Xie, "Hygcn: A gcn accelerator with hybrid architecture", in *2020 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, IEEE, 2020, pp. 15–29.

[234] A. Yimsiriwattana and S. J. Lomonaco Jr, "Generalized ghz states and distributed quantum computing", *arXiv preprint quant-ph/0402148*, 2004.

[235] J. Yu, Z. Wang, V. Vasudevan, L. Yeung, M. Seyedhosseini, and Y. Wu, "Coca: Contrastive captioners are image-text foundation models", *arXiv preprint arXiv:2205.01917*, 2022.

[236] M. Zahran, "Heterogeneous computing: Here to stay: Hardware and software perspectives", *Queue*, vol. 14, no. 6, pp. 31–42, 2016.

[237] S. Zeng, G. Dai, N. Zhang, X. Yang, H. Zhang, Z. Zhu, H. Yang, and Y. Wang, "Serving multi-dnn workloads on fpgas: A coordinated architecture, scheduling, and mapping perspective", *IEEE Transactions on Computers*, vol. 72, no. 5, pp. 1314–1328, 2022.

[238] D. Zhang, S. Huda, E. Songhori, K. Prabhu, Q. Le, A. Goldie, and A. Mirhoseini, "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*, 2022, pp. 27–42.

[239] H. Zhang, K. Yin, A. Wu, H. Shapourian, A. Shabani, and Y. Ding, "Mech: Multi-entry communication highway for superconducting quantum chiplets", in *29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*, 2024.

[240] K. Zhang, H. Ying, H.-N. Dai, L. Li, Y. Peng, K. Guo, and H. Yu, "Compacting deep neural networks for internet of things: Methods and applications", *IEEE Internet of Things Journal*, vol. 8, no. 15, pp. 11 935–11 959, 2021.

[241] T. Zhao, Y. Xie, Y. Wang, J. Cheng, X. Guo, B. Hu, and Y. Chen, "A survey of deep learning on mobile devices: Applications, optimizations, challenges, and research opportunities", *Proceedings of the IEEE*, vol. 110, no. 3, pp. 334–354, 2022.

[242] H. Zou, M. Treinish, K. Hartman, A. Ivrii, and J. Lishman, "Lightsabre: A lightweight and enhanced sabre algorithm", *arXiv preprint arXiv:2409.08368*, 2024.