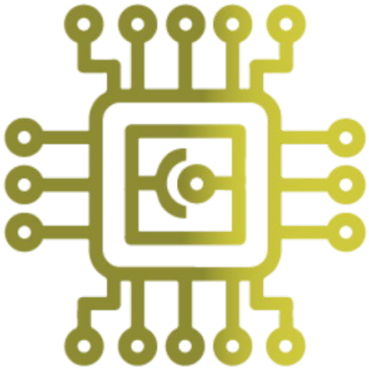


CONVOLVE

Final integrated modelling, design exploration and generation toolflow

GRANT AGREEMENT NUMBER: 101070374

Deliverable D6.3

Description SoC architecture, and the rapid design & prototyping environment

CONVOLVE

Title of the deliverable	Final integrated modelling, design exploration and generation toolflow
WP contributing to the deliverable	WP 6
Task contributing to the deliverable	Task 6.4
Dissemination level	PU – Public
Due submission date	30/09/2029
Actual submission date	30/09/2025
Author(s)	Marian Verhelst, Xiaoling Yi (KUL) Leone Lorenzo, Frank Gurnak (ETHz) Andrea Nardi Dei da Filicaia Dotti (Tue) Alexis Maras, Sotirios Xydis (NTUA)
Internal reviewers	Alejandro Garza (NXP) Tim Güneysu (RUB)

Document Version	Date	Change
V0.1	11/07/2025	Initial version with ToC
V1.0	10/09/2025	Final version

Table of contents

CONVOLVE

Deliverable Executive Summary

This deliverable consolidates the outcomes of Work Package 6 of the CONVOLVE project. WP6 has developed and integrated a comprehensive toolchain for compositional SoC design-space exploration (DSE) and automated system generation, addressing the project's overarching goals of 10× faster design time and 100× energy efficiency improvement for edge AI hardware.

The final toolflow spans the full spectrum from high-level modelling and scheduling to automated hardware generation, SoC integration, and validation. Its key contributions are:

1. Enhanced modelling and scheduling frameworks: Extensions to Stream and ZigZag, leveraging synchronous dataflow (SDF) models and MILP-based scheduling, enable accurate workload mapping and memory-aware scheduling of advanced execution strategies such as line-based layer fusion.
2. Fault tolerance modelling: Integration of redundancy-aware analytical models into Stream allows systematic design-space exploration of selective protection strategies, balancing energy, latency, and reliability for safety-critical domains.
3. Automated hardware generation with VersaCore-SNAX of the optimal design found during design-space exploration: The flexible accelerator generation flow supports dynamic spatial dataflows and efficient integration into SNAX clusters, enabling workload-adaptive AI accelerators.
4. System-level integration of accelerators and validation: Using Bender for dependency management, Morty for RTL flattening, and Chimera for SoC-level AXI integration, the flow enables reproducible builds, automated testing, and rapid SoC prototyping.
5. Praxis framework: A unifying toolchain that links modelling (Stream/ZigZag), hardware generation (VersaCore-SNAX), and compilation (SNAX-MLIR), providing a holistic environment for full-stack design-space exploration.

Together, these advances establish a modular, automated, and validated design methodology that drastically accelerates SoC development compared to the traditional manually-guided flow, while enabling optimized, energy-efficient, and resilient edge AI systems. The deliverable also outlines lessons learned, integration challenges overcome, and opportunities for future extensions of the Praxis framework towards broader workloads and accelerator types.

1.1. Objectives

This document "D6.4 Final integrated modelling, design exploration and generation toolflow" is a deliverable of the Work package No.6 "Compositional architecture DSE and SoC generation".

1.2. WP6 Objectives

WP6 deals with automated compositional system architecture design-space exploration (DSE) and system-on-chip (SoC) generation. Specifically, WP6 focuses on the modular SoC design and rapid deployment which makes the work package one of the contributors to achieve CONVOLVE's target to reduce design time of edge AI hardware systems by 10x by focusing on the faster design time of the SoC architecture and providing a design-space exploration tool for rapid software-hardware co-design explorations. At the same time, WP6 is crucial to bring together all developed accelerators which are needed to achieve CONVOLVE's energy efficiency goals, by providing a SoC template with standard interfaces to a set of ultra-low-power ML and security acceleration blocks which exploit novel architectures, microarchitectures, circuits and devices.

The objectives of WP6 can hence be summarized as follows:

1) Provide a secure and modular RISC-V based SoC architecture template that eases the integration of multiple accelerators, managing control, synchronization, data exchange and run-time reconfiguration.
2) Create a SoC-level performance modelling framework for running ML applications on the targeted modular runtime configurable architectures, integrating the component models coming out of WP2.
3) Develop a rapid design-space Exploration (DSE) framework to cycle quickly over ULP SoC and accelerator constellations, finding the optimal balance between design-time and run-time flexibility.
4) Realize an automated design time instantiation flow for optimal and run-time flexible SoC generation.

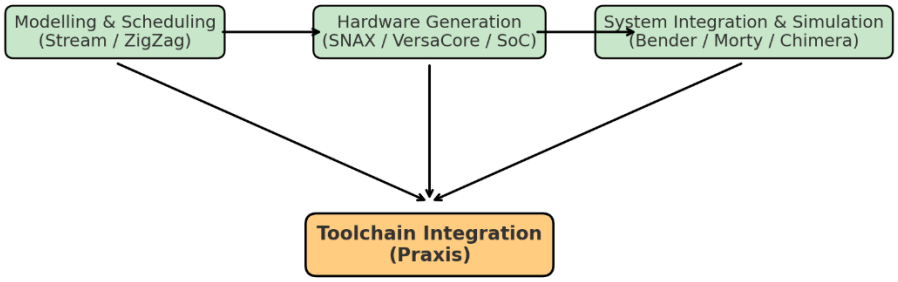
To achieve these goals, it is necessary to have customizable hardware acceleration blocks that can be parameterized during both design and run time using a standard interface. These blocks should allow for various configurations based on diverse application needs. In addition to the RTL design itself, performance models and simulators must also be modifiable to enable fast exploration of the design-space without sacrificing compositional flexibility. WP6 focuses also on automated design-space exploration (DSE) and simulators using performance models of the hardware building blocks.

1.3. Deliverable D6.4 Objectives

Deliverable D6.4 is the final output of Work Package 6. The objective of this deliverable is to consolidate and document the full toolchain developed within WP6, spanning from high-level modelling and performance estimation to automated hardware generation, integration, and validation.

Specifically, this deliverable aims to:

- Capturing the final enhancements made to the modelling frameworks Stream and ZigZag, and their usage towards for accurate system-level scheduling and workload mapping, including integration with synchronous dataflow (SDF) models and MILP-based scheduling.
- Present the automated hardware generation flow via the SNAX, VersaCore and SoC-level platforms, capable of generating flexible accelerator clusters and complete multi-cluster systems-on-chip. based on different dataflow configurations and supporting both design-time and runtime parameterization.
- Describe the system integration and simulation flow that enables streamlined SoC-level prototyping and validation, highlighting the automation infrastructure (e.g. Bender, Morty, Chimera) developed to manage IP integration, RTL hierarchy, and build processes.
- Showcase the opportunities stemming from further toolchain integration via the Praxis toolflow, linking high-level modelling and scheduling tools (Stream/ZigZag) with hardware generation (VersaCore-SNAX) and compiler flows, enabling a modular and automated path from model to silicon.



Overall, this deliverable serves to validate that the WP6 toolflow achieves its goal of drastically reducing design time for modular SoC-based edge AI systems while enabling optimized energy-efficient solutions through tight software-hardware co-design.

1.4. Deliverable D6.4 Structure

This deliverable is structured to reflect the integration of all components developed in WP6, following a logical flow from modelling and scheduling to hardware generation, combined tools integration and final use-case validation:

- Section 2 describes the enhanced performance modelling and scheduling framework, including SDF-based modelling, MILP scheduling, and integration into the Stream/ZigZag toolset.
- Section 3 presents the automated hardware generation flow at the accelerator and system level, and details the system integration, simulation, and verification infrastructure developed within the project.
- Section 4 outlines the full toolchain integration in the Praxis flow, connecting high-level modelling to hardware generation and compiler support.
- Section 5 summarizes the achievements of WP6, reflects on lessons learned, and outlines directions for future development.

2. Enhancing Stream for efficient performance modelling, workload mapping and scheduling exploration

Layer-fusion (aka depth-first or cascaded execution), is the interleaved execution of convolutional layers. The advantages of this method of execution are twofold. Firstly, this method reduces the required memory of execution, allowing for a smaller on-chip memory footprint and fewer accesses to off-chip memory. Secondly, this method allows the hiding of both computation and data-transfers through pipelining, and maximizes parallelism and utilization of processors and communication channels.

To support the periodic scheduling of line-based layer-fusion, we will discuss the mathematical modelling of the data-dependency of line-based layer-fusion as the formalism of periodically dependent tasks, Synchronous Dataflow Graph (SDF). This is a continuation from the work described in deliverable 6.2, with here, a specific focus on integration with the DSE framework Stream. The framework has further been extended to allow a compact schedule representation, which experimental results are described in deliverable D5.3.

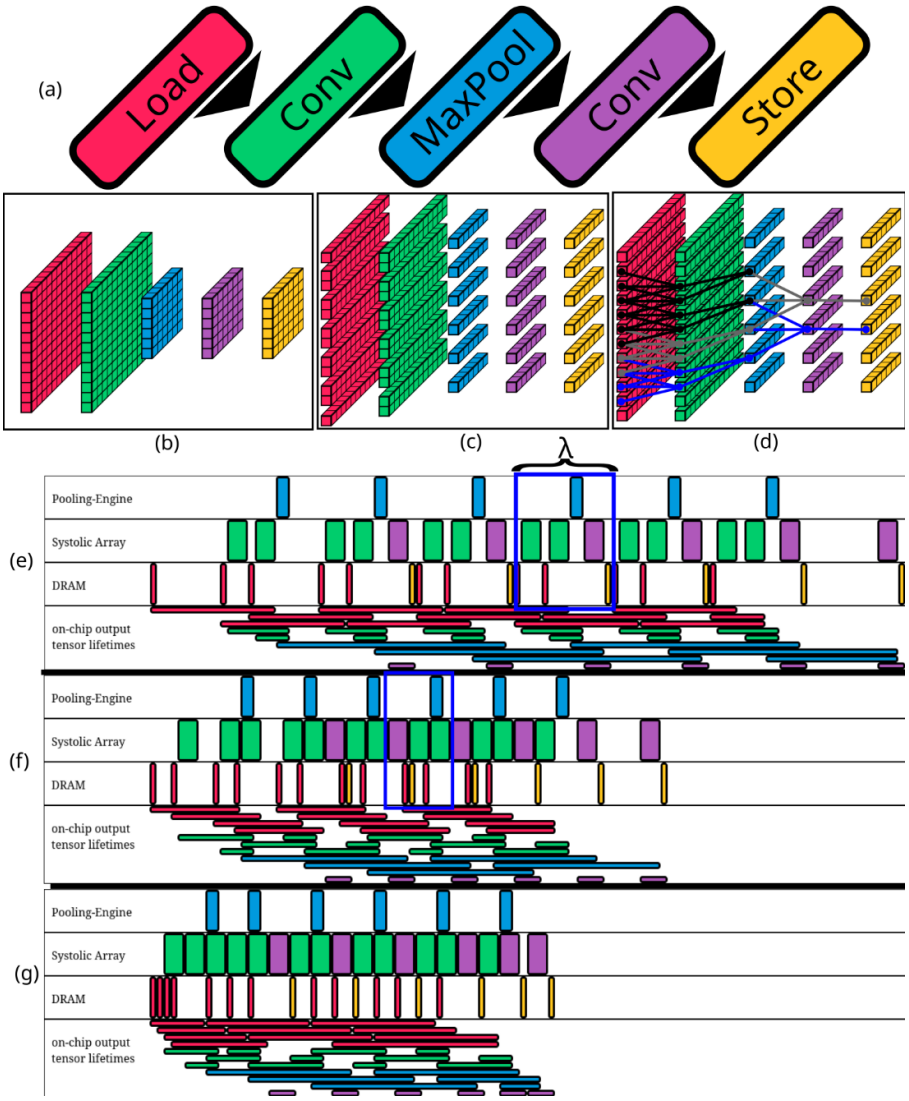


FIGURE 1: LINE-BASED LAYER FUSED SCHEDULING

In this work we will model line-based layer-fusion, and we will be referring to a "line", the row or column of a tensor. One such example of line-based layer fused execution is presented in Figure 1. Because layer-fusion is the interleaving of both computation (i.e. convolutions) and data transfers, and because we model data-dependence between both computation and data-transfers, we will refer to both computation and data transfer by the term "process".

2.1. Formulation of periodic dependence of line-based layer-fusion as an SDF

To model the data-dependency of processes in a line-based layer-fused execution, for each process x, we will define, for each process, a function for stride ($str(x)$), padding ($pad(x)$) and window-size ($win(x)$) in the direction of the split of the convolution. With padding, we only consider padding that affects the execution of the first few lines. To give an example, we model a 3x3 convolution with stride 2 and padding (1, 1, 1, 1) as a process of stride, padding and window-size of respectively $str(x)=2$, $pad(x)=1$ and $win(x)=3$. The first execution of a process can occur only after $win(x)-pad(x)$ input lines are available, and its subsequent executions occur every additional $str(x)$ input lines are available.

We can model such periodic behaviour as a Synchronous Dataflow Graph (SDF). To give an example of such modelling, we use, the example of Figure 1. In our example the processes are respectively Load, $Conv_1$, MaxPool, $Conv_2$ and Store as shown in Figure 1. For each process x, their consumption rate is of $str(x)$ (as each subsequent sub-convolution requires an additional $str(x)$ lines to execute) and $str(x) - win(x) + pad(x)$ initial tokens to in its input channels (as it requires $win(x)-pad(x)$ initial lines for its initial execution).

2.2. Augmentation of the SDF for architectural constraints

In the previous section, we have discussed how to model the periodic data-dependence of line-based layer-fusion as an SDF. While so, they don't model architectural constraints of the accelerator on which they are being executed on. Two of the architectural characteristics that are important being, memory bound and contention. What we mean by memory bound, is the memory occupancy of a memory may not surpass its memory capacity. What we mean by contention, is that, if a process acquires exclusive access to a hardware resource (i.e. a processor or a communication channel), the execution of two processes that acquire such same hardware resource may not overlap in time.

2.3. Modelling memory occupancy

To model memory occupancy, we use the SDF notion of fixed capacity FIFOs [1]. We augment the notion of fixed size FIFO with the addition of a ``free" actor f_i of execution time 0, the firings of which represent the moments when the output tensor of a process stops having dependents, and it is safe to be freed. This is used in simulation to determine when in time a tensor is freed. We add a channel from the free actor to the process actor, with initial tokens b_i, which represents the maximum number of outputs of the process (which are lines of the activation tensor) that are allowed to be live at any point in time. We model memory occupancy as the weighted sum of the capacities of the FIFOs the memory accommodates, with such

[1] Moreira, Orlando, et al. "Buffer sizing for rate-optimal single-rate data-flow scheduling revisited." *IEEE Transactions on Computers* 59.2 (2009): 188-201.

weight being the size of the output tensor line the FIFO accommodates. As an example, in Figure 1, in schedule (e), the input lines loaded from off-chip are stored in a FIFO of capacity 3, while in schedule (f) and (g), the input lines loaded from off-chip are stored in a FIFO of capacity 5. Given that the loaded input line is of size 12 bytes (assuming numerical precision of 8 bit integers), the memory occupancy due to input lines from off-chip is of 36 bytes for schedule (e) and 60 bytes for schedule (f)-(g). Because we are only interested in the memory occupancy and not directly in the FIFO capacities, we assign variable capacities b_i which will be determined at scheduling time.

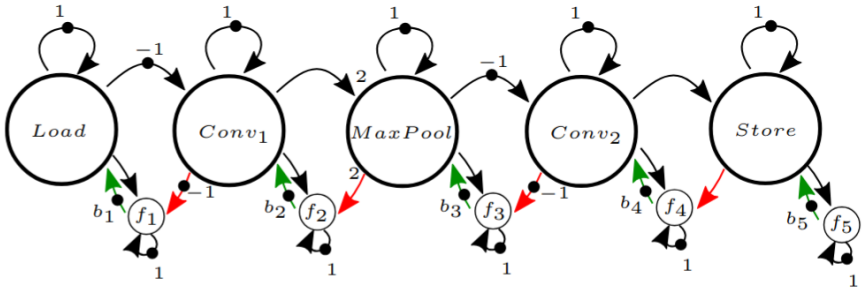


FIGURE 2: WORKLOAD MODELLED AS AN SDF.

2.4. Modelling contention and non-overlap

To model non-overlap among processes that acquire the same hardware resource, we use the concept of disjunctive arcs, used in the literature of recurrent job-shop [2]. We apply, what in the literature is often described as an HSDF expansion. Given the HSDF expansion, we add the disjunctive arcs, as shown in Figure 3 with blue channels.

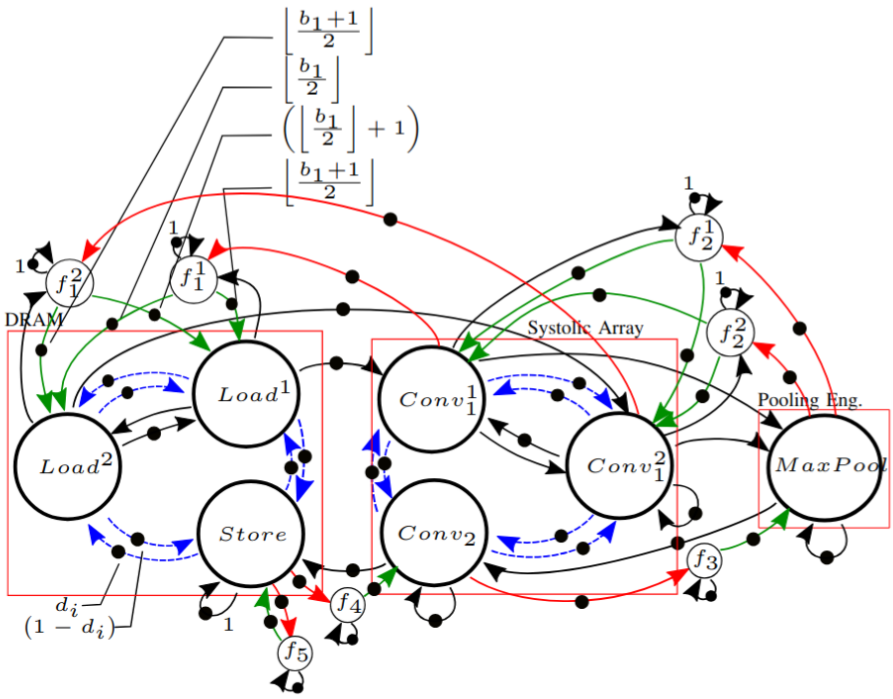


FIGURE 3: HSD F EXPANSION, WE ADD THE DISJUNCTIVE ARCS

[2] Hanen, Claire. "Study of a NP-hard cyclic scheduling problem: The recurrent job-shop." *European journal of operational research* 72.1 (1994): 82-101.

2.5. MILP formulation

Given the SDF in Figure 3, we can use a MILP to determine the FIFO capacities and the order of execution of the processes. We formulated the MILP to optimize for throughput, with one such instance of execution, being displayed in Figure 4, and the structure of the executed CNN, its prologue, steady state and epilogue.

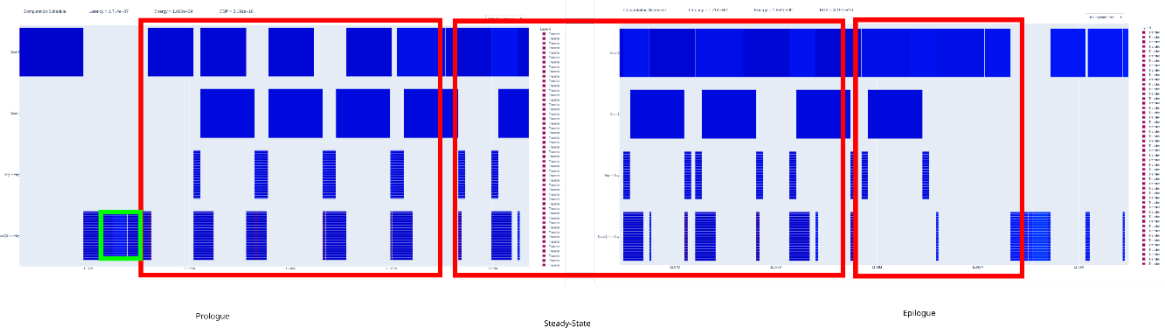


FIGURE 4: PROLOGUE, STEADY-STATE AND EPILOGUE OF THE EXECUTION SCHEUDLE

2.6. Required Modifications to the Stream scheduling framework

We implemented our scheduling extension to the DSE tool Stream. The version of Stream we extended is of git hash 31f6f77, which internally assumes ZigZag version 3.2.0.

1. Stream explicitly instantiates the sub-convolutional operations and, through an r-tree data structure, fetches their dependencies. This is not required if we generate an SDF as in Figure 2, as the interdependence is modelled for. That said, the number of executions of the sub-convolutions should also be stored to limit the number of executions in simulation. Using an SDF for the internal representation of a network or stack will require modifications of the processing by Stream, which, instead of processing the entirety of the workload as a directed acyclic graph with interdependence between the sub-convolutions, will only require the processing of the SDF.
    - Stream uses a separate scheduling logic for computation and data transfer. Computations of Stream are handled by StreamCostModelEvaluation and data transfers are handled by CommunicationManager. Unfortunately, there is no simple way to extend the scheduler to allow a static schedule of both computation and data-transfers. To validate that the generated schedule is a valid schedule under Stream's semantics of execution, we split scheduling and simulation to be done independently. In the scheduling phase we generate a schedule under SDF semantics.
    - In simulation, we use the Stream's lower-level function calls of MemoryManager, such as:
        - add_tensor_to_core
        - remove_tensor_from_top_instance
        - get_memory_energy_cost_of_transfer
        - block_offchip_links

This, to reproduce the schedule, and populate StreamCostModelEvaluation and CommunicationManager with the events required for analysis and verification of the schedule. This is done also to ensure that the scheduling of the SDF respects executional semantics of Stream, the required tensors are available within the memory of the core at the correct time, and to ensure that the memory occupancy is within memory size.

2.7. Comparison With State-of-the-Art

We conducted experiments with a heterogeneous multi-core system in Figure 5, with the memory hierarchy described in Table.

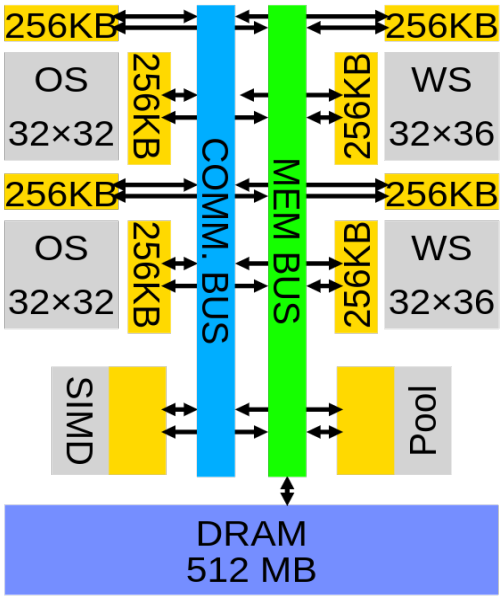


FIGURE 5: ARCHITECTURE FOR WHICH EXPERIMENTAL EVALUATION WAS CONDUCTED

Core	MAC array	Spatial Unrolling	Reg. per MAC	Local Memory	Activation Memory	Weight Memory
OS	32 × 32	K: 32 OY: 32	O: 4B	O: 4KB	I,O: 256KB	W: 256KB
WS	32 × 36	K: 32 FX: 3 FY:3 C:4	O: 4B	O: 128B × 32	I,O: 256KB	W:256KB
SIMD	64	K:64	NA		I,W,O:128 KB	NA
Pooling Engine	2×2×8	FX: 2 FY:2 K:8	NA		I,W,O:128 KB	NA

TABLE 1: MEMORY HIERARCHY OF EACH CORE FOR THE ARCHITECTURE IN FIGURE 4

As it can be seen in Table 2, our method is competitive in latency and energy. While so, our modelling of memory occupancy as fixed capacity FIFOs allows us to consider fragmentation (shown in the table as activation memory distributed). Further, because we determine memory at scheduling time, this method does not suffer from, a not required, memory overutilization. Even without considering memory fragmentation, our method can determine a more realistic memory utilization without major deterioration of latency.

	ResNet18		FSRCNN		MobileNetV2		Xception	
	Sota	Our	Sota	Our	Sota	Our	Sota	Our
Latency[10^6cc]	14.37	17.82	38.62	38.63	3.63	3.34	71.28	66.70
Energy [mJ]	8.92	8.28	10.82	10.81	1.29	1.11	29.03	29.39
Activation Memory Unified [KB] (Sum)	311.64	106.70	511.17	137.66	493.99	98.67	969.31	466.82
Activation Memory Distributed [KB] (Sum)	NA	113.70	NA	162.97	NA	118.35	NA	568.00

TABLE 2: NUMBERS OF LATENCY, ENERGY, ACTIVATION MEMORY UNIFIED(I.E. MEMORY OCCUPANCY DISREGARDING FRAGMENTATION) AND ACTIVATION MEMORY DISTRIBUTED(I.E. HAD THE TENSOR BEEN STORED IN CIRCULAR BUFFERS, SUM OF THE MEMORY ALLOCATED PER CIRCULAR BUFFERS, AND THEREFORE, FRAGMENTATION AWARE). THE MEMORY NUMBERS ARE THE SUM OF THE MEMORY NUMBERS AMONG THE CORES.

2.8. Conclusion

In this section, we have shown how we model the periodic data-dependency of line-based layer fusion as an SDF, how to optimize the schedule under memory bound and contention and the modifications required for the integration with Stream. We experimentally showed that this methodology of scheduling allows significantly smaller memory utilization compared to the SotA, and allows consideration of fragmentation. We further discuss experimental results in the deliverable D5.3 for the compressing of the schedule representation of line-based layer-fusion.

3. Extending Design-space Exploration towards fault tolerance awareness

3.1. Introduction

Design-space exploration (DSE) tools such as ZigZag and Stream are primarily employed to optimize the mapping and scheduling of DNN kernels on single- and multi-core accelerators. However, their analytical capabilities extend further, enabling seamless integration into frameworks that target architectural optimization or model-level modifications. Leveraging the accurate analytical models provided by these tools, we aim to utilize them for assessing the costs and overheads introduced by modular redundancy methodologies when applied to DNN architectures. This integration will allow us to quantitatively evaluate the trade-offs involved in ensuring reliable inference and enhancing resilience against soft errors.

Single Event Upsets (SEUs) are a major reliability concern in advanced technology nodes and low-power conditions. Caused by high-energy particle strikes that deposit localized charge, SEUs can invert the state of memory or logic elements, appearing as bit-flips or glitches that

may propagate and result in silent data corruption (SDC) [R1]. While rare under nominal conditions ($\approx 10^{-6}$ fault rates), aggressive techniques such as voltage scaling can raise error rates to $\approx 10^{-2}$ [R2]. Large-scale studies further report SDCs occurring as often as once per several thousand cores [R3]. The need for effective mitigation is especially critical in safety-critical domains (e.g., automotive, medical), where undetected errors can have severe consequences. Deliverables 2.6 and 2.7 demonstrate how injected errors in DNN parameters and activations degrade reliability, underscoring the necessity of mitigation strategies for dependable inference on edge accelerators.

Traditional techniques such as Dual Modular Redundancy (DMR) and Triple Modular Redundancy (TMR) are highly effective in protecting systems against soft errors. However, their reliance on spatial and temporal replication introduces significant overheads, making them impractical for low-power or resource-constrained applications. An alternative strategy builds on the observation that not all parameters and activations contribute equally to the final inference outcome. By selectively identifying and protecting the most critical components, it is possible to maintain high model accuracy while significantly reducing overheads in memory, latency, and energy consumption. Prior works [R4] – [R7] investigate these trade-offs by applying redundancy at different levels of granularity, ranging from entire layers to finer structures such as kernels, filters, or even individual neurons. However, these approaches do not incorporate analytical modelling tools into their methodologies. This omission restricts their ability to:

- systematically explore trade-offs across different levels of protection and
- evaluate the influence of architectural design choices on the overheads introduced by modular redundancy.

In this line of work, by leveraging Stream, we enable a systematic and fine-grained per-layer analysis, supporting not only varying protection levels (e.g., percentages of kernels or neurons) but also the application of different replication strategies (such as DMR or TMR) across layers. A key distinction of our approach is the explicit integration of latency and energy costs of replication into the optimization process, ensuring a balanced trade-off between resilience and efficiency. Beyond single-core settings, we extend this exploration to multi-core architectures, demonstrating how architectural factors, such as processing element array dimensions, and the number of cores, influence the effectiveness of selective protection strategies.

3.2. Impact of Soft Errors in DNN accuracy

Soft errors most commonly manifest as single-bit flips, where a bit stored in memory elements such as SRAM, registers, or flip-flops is inverted from 0 to 1 or vice versa. As a result, bit-flip faults are widely adopted to model soft errors in fault injection simulations [R8]. Such errors can corrupt data stored in memory and propagate through the hardware data flow, thereby affecting subsequent operations and ultimately leading to incorrect computations. In deep learning models, this can translate into a significant loss of prediction accuracy. Following the example of those works, we decide to model soft errors as random bit-flips applied to parameters (weights and activations) during inference. Reported results are averaged across

10 independent experiments for each model and Bit Error Rate (BER) configuration to ensure statistical reliability.

Figure 6 illustrates the behaviour of the ResNet-18 and ResNet-50 architectures, evaluated on the CIFAR-10 and ImageNet datasets, respectively, under different precision formats in the presence of soft errors. The results clearly indicate that quantized models exhibit higher robustness compared to their floating-point counterparts. This outcome is expected, as floating-point models require larger memory footprints, making them more susceptible to errors at a given fault rate. Moreover, bit flips in critical fields such as the exponent or sign bit can cause disproportionately large deviations from the original values, explaining the sharp fluctuations observed in their accuracy. In contrast, quantized integer formats constrain error propagation, leading to more localized and bounded deviations.

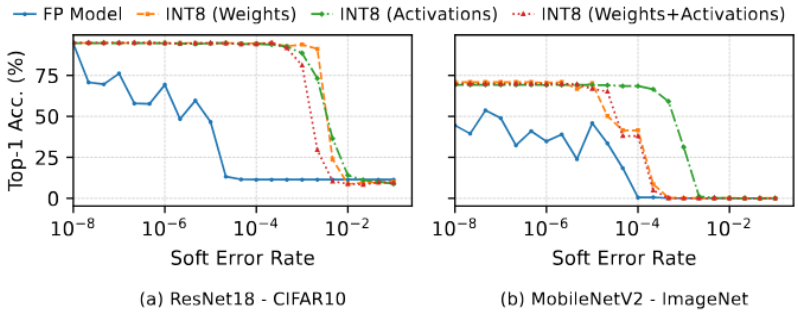


FIGURE 6: PERFORMANCE OF RESNET-18 AND RESNET-50 UNDER FP32 AND 8-BIT QUANTIZATION WHEN ERRORS ARE INJECTED INTO THE ENTIRE NETWORKS AND ON ISOLATED STRUCTURES (WEIGHTS OR ACTIVATIONS FOR THE QUANTIZED MODEL)

It is important to note that not all layers in a DNN exhibit the same resilience to soft errors, as shown in Figure 7. While these characteristics may vary depending on the model architecture and the classification task, this observation highlights the benefit of applying selective protection, reinforcing the most vulnerable layers while allowing others to remain less protected. We also report the percentage of kernels that must be protected to limit accuracy degradation to less than 1%. The results reveal a strong correlation between a layer's sensitivity and the required level of protection. Therefore, rather than safeguarding all parameters, selectively protecting only the most influential ones can ensure robust inference with minimal overhead.

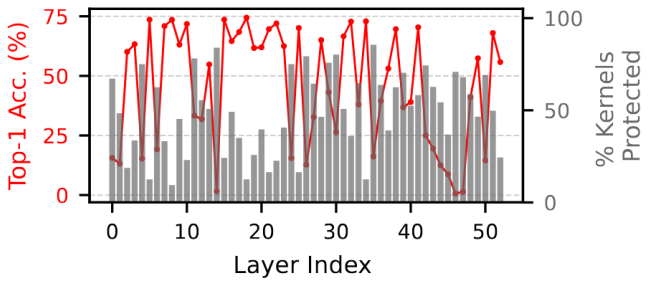


FIGURE 7: LAYER-WISE ANALYSIS OF RESNET-50 (IMAGENET) UNDER A BER OF 10–4, SHOWING ACCURACY DEGRADATION (RED LINE) AND THE MINIMUM PERCENTAGE OF KERNELS (BAR PLOTS) THAT MUST BE PROTECTED IN EACH LAYER TO MAINTAIN ACCURACY.

3.3. Methodology Overview

An abstract overview of the proposed methodology is presented in Figure 8. We formulate the problem as a multi-objective optimization task, seeking to minimize accuracy degradation through modular redundancy while simultaneously constraining the energy and latency overheads introduced by replication. The approach selectively replicates the most sensitive structures of DNN models (e.g., output neurons or activation feature maps) to maximize resilience where it is most impactful. At each iteration, the framework identifies the minimum set of parameters requiring protection within each layer. Depending on the layer's vulnerability, either DMR or TMR is applied. After injecting errors into both weights and activations during inference, the model's accuracy is evaluated. To complement this, we employ Stream to estimate the corresponding latency and energy costs, while also exploiting its ability to optimize dataflow to mitigate the cost of the redundant computations and memory transactions.

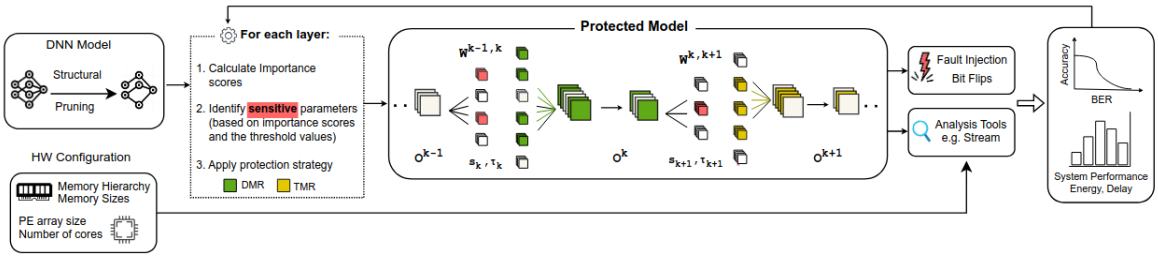


FIGURE 8: OVERVIEW OF THE PROPOSED WORKFLOW

3.4. Selective Structural Pruning

To ensure that both weights and activations are protected during inference, we apply replication at the granularity of each layer's outputs. Specifically, for convolutional and fully connected layers, we replicate in memory the weights associated with the most sensitive output feature maps and neurons. In this way, weights gain protection through multiple storage copies, while the corresponding outputs are safeguarded by being redundantly recomputed during inference.

To identify which parameters, warrant protection, we employ the weight-sum method, a technique widely used in DNN compression. In our context, this metric highlights the neurons or kernels that contribute most significantly to a layer's output and are thus more vulnerable to soft errors, since faults in these parameters are more likely to cause broader accuracy degradation [R6]. By ranking parameters according to their importance and selectively safeguarding those with the highest scores, our approach ensures that redundancy is applied where it is most critical, minimizing overhead while preserving robustness. The score of each neuron or kernel in each layer is calculated as the L1-norm of the weights leading up to it:

$$s_l^i = \sum_{j=1}^{C_{l+1}} \left| W_{i,j}^{(l,l+1)} \right|$$

This method offers two key advantages. First, it effectively captures parameter importance, delivering results comparable to more sophisticated techniques such as second-order derivative or entropy-based methods [R6]. Second, its simplicity enables rapid design-space exploration (DSE) and efficient evaluation within our multi-objective framework, unlike more computationally intensive alternatives.

3.5. Heterogeneous Modular Redundancy

A key design decision lies in determining the level of redundancy to apply per layer (DMR vs. TMR) and defining the strategy for resolving discrepancies when redundant outputs differ. While TMR naturally offers stronger and more reliable protection, it also introduces substantially higher latency and energy overheads. Furthermore, as illustrated in Figure 7, not all layers demand the same degree of fault tolerance. In many cases, applying DMR selectively (e.g., to only a subset of kernels) is sufficient to preserve accuracy [R7]. These observations motivate the development of selective and less aggressive protection methodologies that balance reliability and efficiency, ensuring robustness where it is most critical without incurring unnecessary cost.

Figure 9 illustrates the efficiency of widely used modular redundancy methodologies under full-network error injection, assuming that all parameters are protected. For TMR, we evaluate both a median-based strategy and a majority voter that defaults to the minimum when all outputs differ. For DMR, we consider selecting either the minimum or the average of the two values. Results show that the median filter in TMR and the mean-based strategy in DMR (slightly better than picking the minimum value) are the most effective within their respective approaches and are therefore adopted in the remainder of this work. However, for the more challenging ImageNet classification task, DMR alone fails to achieve the same level of robustness as TMR. In such cases, a hybrid solution that strategically combines DMR and TMR offers a practical means of balancing protection and overhead.

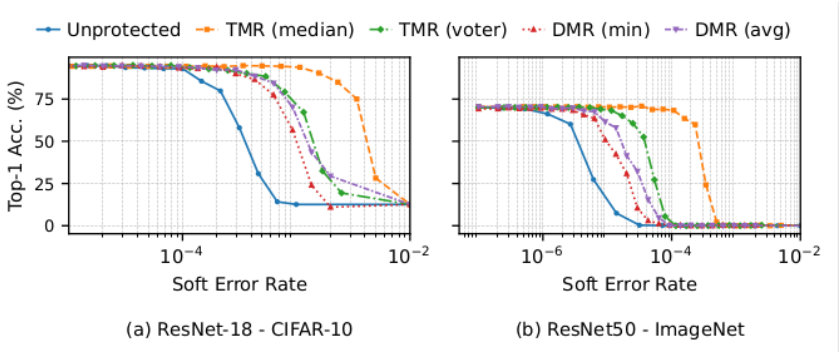


FIGURE 9: PERFORMANCE COMPARISON OF MODULAR REDUNDANCY METHODOLOGIES.

3.6. Multi-objective Optimization

At this stage, our primary objective is to determine, for each layer of the network, the optimal threshold that dictates which kernels are replicated and which remain unprotected, along with the most suitable protection strategy. These design choices directly shape the trade-off between reliability and efficiency: while higher protection levels improve accuracy under soft

errors, they also introduce additional MAC operations and data transfers between main memory and the accelerator(s), thereby increasing both latency and energy consumption.

The threshold for each layer is defined as the mean importance score reduced by a scaled standard deviation. Kernels or neurons with scores above the threshold value are selected for replication, while those below remain unprotected. This formulation enables a fine-grained balance, ensuring that redundancy is applied selectively to the most critical structures while minimizing unnecessary overhead. More specifically, the threshold value is set as following:

$$\tau_l = \mu_l - k \cdot \sigma_l$$

where μl and σl denote the mean and standard deviation of the importance scores in layer l. The parameter k serves as a tunable design variable within our multi-objective optimization framework. Higher values of k yield stricter thresholds, resulting in fewer parameters being replicated and thus lower overhead. Conversely, smaller values of k relax the threshold, expanding protection coverage by replicating more parameters and thereby enhancing resilience. Intuitively, layers more sensitive to soft errors correspond to lower threshold values, while more resilient layers are assigned higher thresholds.

Optimizing the set of the k values across the network, along with selecting DMR or TMR for each layer, enables an efficient exploration of a large design-space. Since our objective is not only to enhance resilience but also to mitigate the computational and memory overheads introduced by redundancy, we rely on Stream, owing to its ability to accurately model the performance of both single-core and multi-core systems, while simultaneously optimizing mapping and dataflow for efficient DNN inference. To capture the overheads of our selective protection scheme, the protected version of the network, reflecting the number of replicated parameters and the assigned protection strategy per layer in each iteration, is provided as input to Stream. From this, we obtain accurate latency and energy estimates. This formulation ensures that both threshold selection and redundancy strategy assignment are guided not only by each layer's fault sensitivity, but also by its contribution to the latency and energy profile of the system.

3.7. Evaluation & Discussion

3.7.1. Design-space Exploration Analysis

We evaluate the efficiency of our methodology by analysing the DSE results across a diverse set of architectures and datasets, including ResNet-18 on CIFAR-10, VGG16 on CIFAR100, and ResNet-50 on ImageNet. To guide the optimization, we employ the NSGA-II algorithm, enabling efficient exploration of the large search space while maintaining a diverse set of trade-off solutions that balance accuracy and system overhead. For this analysis, the hardware platform selected isa single-core TPU-like accelerator featuring a 128 × 128 systolic array (the preferred size in Google's TPUv4), a 2 MB SRAM for weights, inputs, and intermediate results, and an off-chip DRAM interface for bulk transfers.

In our DSE experiments, shown in Figure 10, we evaluate 200 candidate solutions for each benchmark to ensure sufficient diversity while keeping runtime practical. In addition to the DSE results, we also report the accuracy of the solution chosen near the knee of the Pareto

front, as it offers the best balance between accuracy preservation and system-level overhead. The accuracy of these selected solutions across different soft error rates is also reported and highlighted in orange in the scatter plots. Our analysis demonstrates that by selectively protecting only the most critical kernels, it is possible to achieve resilience levels comparable to those of fully protected models, but at only a fraction of the cost. Compared to full TMR, our approach identifies Pareto-optimal solutions that provide similar robustness while improving system performance by up to 1.77× in energy and 1.95× in latency, although still incurring average overheads (EDP) of up to 44% relative to the unprotected solution.

These results also underscore the importance of integrating system-level metrics into the optimization process (especially when taken from analytical modelling tools like Stream). When redundancy is evaluated without considering performance metrics, solutions may appear competitive in terms of accuracy but fail to account for energy and latency implications, ultimately leading to sub-optimal trade-offs. Similarly, relying on coarser-grained overhead estimations, such as simply counting MAC operations or memory accesses, can result in substantial misestimations. For example, while TMR may theoretically triple the number of MAC operations, the actual overhead can differ significantly, as illustrated in Figure 10, depending on tensor dimensions, architectural characteristics, and how effectively the hardware resources are utilized.

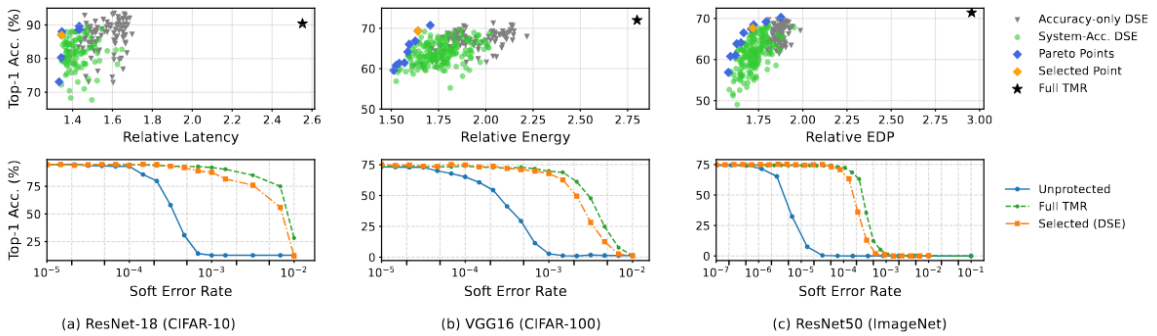


FIGURE 10: IMPACT ASSESSMENT OF THE PROPOSED METHODOLOGY ACROSS 3 DIFFERENT NEURAL NETWORKS, RESULTS OBTAINED THROUGH SIMULATION WITH STREAM.

To meet the stringent demands of modern applications, most SoCs now integrate multiple accelerators, either homogeneous or heterogeneous, ranging from 2 [R9] up to even 16 cores [R10]. This increased parallelism can prove beneficial for our approach, as replicated kernels have no dependencies and can therefore be executed independently across different cores. Although direct measurements on hardware [R5] provide higher accuracy, they limit design-time exploration by preventing evaluation of alternative architectural configurations, which is essential for jointly optimizing reliability and performance. Experimental results with Stream (Figure 10) show that the relative EDP overheads compared to the unprotected baseline drop sharply as the number of DNN accelerators increases from an average of 44% on a single core, down to 30% with 2 cores, 18% with 4 cores, 10% with 8 cores, and just 4% with 16 cores. These results demonstrate that redundancy policies are particularly well-suited for multi-core systems and, when combined with efficient dataflow strategies, can be integrated into safety-critical domains without incurring significant overheads.

3.8. Conclusion

In this section, we introduced a platform-aware methodology to enhance the robustness of DNNs against soft errors through modular redundancy, while explicitly accounting for energy and latency overheads. We formulated the problem as a multi-objective optimization task, selectively protecting only the most critical parameters to minimize replication while preserving high accuracy in error-prone environments. Our results highlight the importance of analytical modelling tools such as Stream for accurately assessing the overheads of redundancy strategies and underscore the need for co-optimizing resilience and system performance. Finally, we demonstrate that redundancy policies are particularly well-suited for multi-core architectures, where parallelism significantly reduces their relative overhead. These results contribute to Objective 4 of Convolve - Smart Edge Processing enabling mechanisms. This method improves reliability by redundancy without sacrificing the energy consumption improvements.

4. Automated hardware generation of versatile accelerator clusters with Versacore

4.1. SNAX Cluster

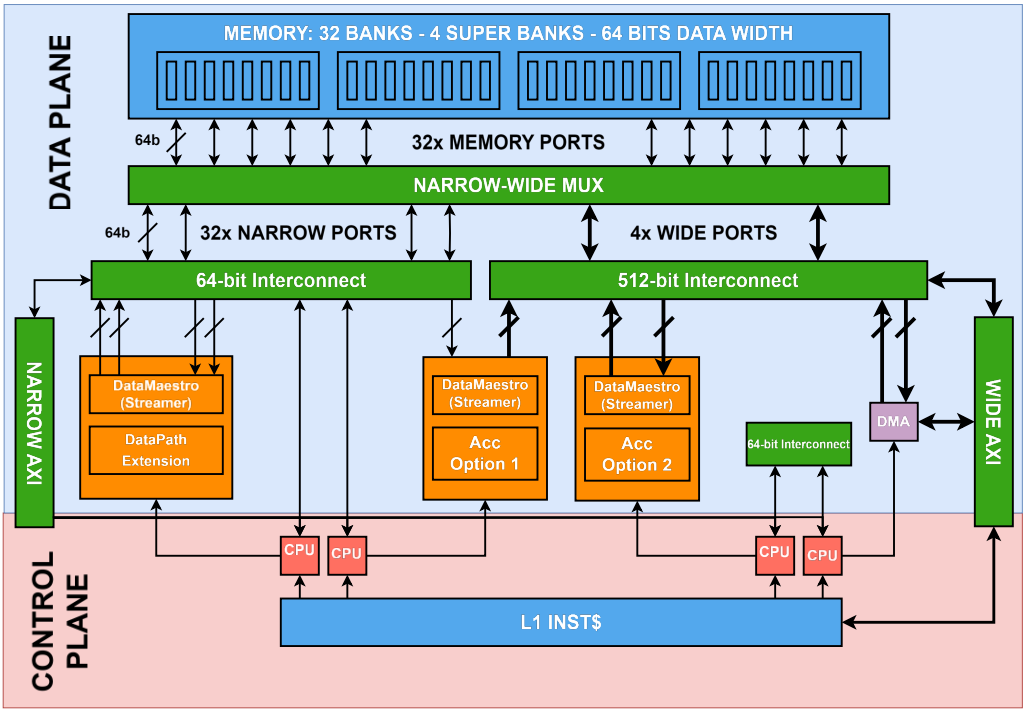


Figure 11: SNAX cluster overview

The SNAX cluster (Figure 11) is the main system that integrates all accelerators and has all the peripherals added to it. It contains the main memory, a complex TCDM interconnect, Snitch cores for controlling the accelerators, an instruction cache that is shared with all the Snitch cores, a DMA to transfer data from an external memory and into the TCDM memory, and AXI interconnects for communicating outwards of the cluster.

To support agile accelerator integration, SNAX employs a hybrid data/control coupling strategy to boost the accelerator's ease of programmability and efficiency at the system level. A loosely coupled control interface enables efficient kernel offloading without stalling the RISC-V Snitch core, while the CSR-based configuration model abstracts hardware details for ease of programming. In parallel, tightly coupled data streamers provide low-latency access to shared memory, accommodating diverse bandwidth demands at design time and a variety of data access patterns at runtime for diverse workloads.

It is worth noting that there are two kinds of TCDM interconnects: a narrow interconnect that has a low bandwidth (64 bits per port) and a wide interconnect that has a higher bandwidth (512 bits per port). The narrow interconnect has fine-grained access to the memory but has higher interconnect complexity as the number of accelerator or core ports increases. The wide interconnect has a higher granularity of access, making it less flexible since it needs to access

data in super banks (as indicated by the sub boxes in the memory). The wide interconnect, however, has a smaller number of ports to manage and therefore has less circuit complexity.

4.2. The VersaCore architecture

The VersaCore is a spatial computing array generator that can dynamically reshape the array at compile time according to the dimensions specified at generation time. We will first explain the spatial dataflow supported by VersaCore in more detail. Figure 12 illustrates the conceptual generation process of VersaCore, covering three main stages: the arithmetic unit generation, array shape construction, and spatial and temporal data reduction. Firstly, as shown in Figure 12, VersaCore generates many multipliers and adders in parallel, building the basic arithmetic units in the spatial array. Secondly, these multipliers are dynamically organized towards different shapes to support various spatial data reuse patterns. For example, Figure 12(a1) and (a2) shows a multiplier array arranged in a 1-D manner to maximally support scalar data broadcast. In contrast, Figure 12 (c1) and (c2) shows a multiplier array organized in a 3-D fashion to maximally support 2-dimensional vector data broadcasts. VersaCore supports arranging the multipliers into an arbitrary dimension array and supports N-D spatial data broadcast patterns. Lastly, the multiplier outputs are collected in a reduction stage, formed by the adders. All kinds of temporal and spatial data reduction can be applied. As shown in Figure 12 (b1) and (b2), a 2D array can conduct temporal reduction for every output of the multipliers or conduct spatial reduction across one dimension, facilitating spatial data reuse before storing the result into the storage units. Figure 12 (c1) and (c2) shows a scenario where a shallow and deep spatial reduction is conducted in the 3D array. VersaCore supports organizing the multiplier array and conducting spatial reduction in an N-D manner, facilitating N-D spatial dataflow. More importantly, different spatial dataflows can be hosted in one set of arithmetic units substrate, and the spatial dataflow can be dynamically changed to achieve the best spatial utilization under diverse workload characteristics.

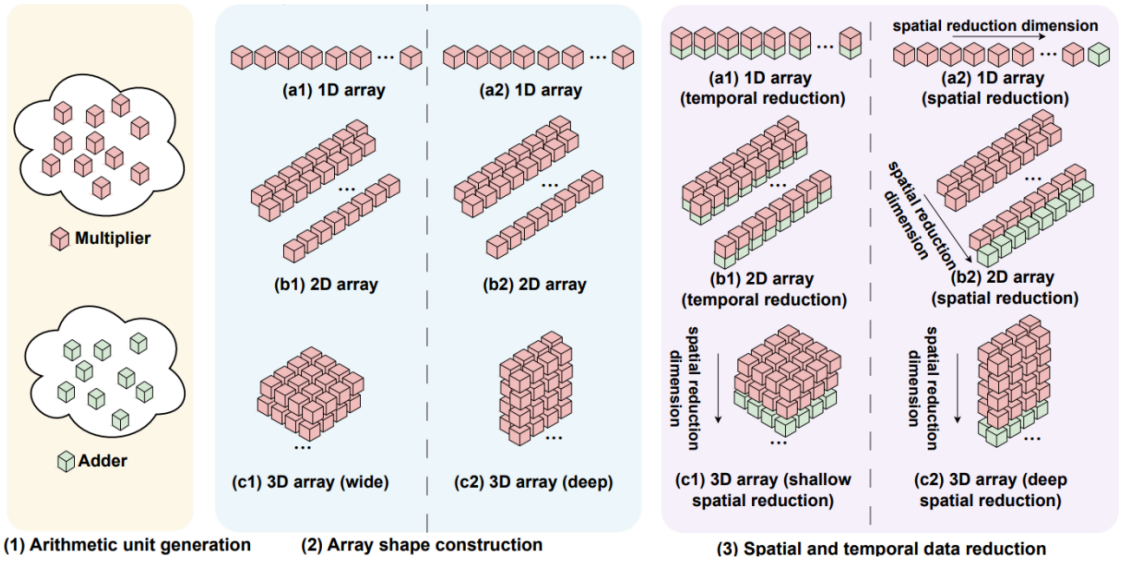


FIGURE 12: VERSACORE DYNAMIC SPATIAL DATAFLOW CONCEPT

Figure 13 illustrates the architecture of VersaCore, which comprises several key components, including data gather and scatter, data dispatch network, multipliers, adder tree, accumulator, and array top controller. Within VersaCore, data flows exclusively from top to bottom. To support architecture generation with dynamic dataflow and data type, VersaCore equips highly

configurable parameters for different blocks during generation and compile time. Below, each component is introduced in depth.

1) Data Gather and Scatter: When a data operand is stationary, it can be reused across cycles without accessing memory each time. In VersaCore, we use data gather and scatter to support splitting a wide stationary operand access port into several narrower ports, i.e., taking in/sending out the stationary operand in several cycles instead of one cycle, to save DataMaestro's memory access channel count and reduce bank conflicts, without sacrificing any performance. VersaCore has three data gathers for the input, weight, and partial sum ports, and a scatter for the output port. For example, when in the weight stationary mode, the weight data gather will gradually store the sliced data from the DataMaestro in different cycles. When in the output stationary mode, the output scatter will send out the stored wide output data piece by piece to the DataMaestro. When a specific operand is stationary, the data gather/scatter will be enabled; otherwise, it will be disabled.

2) Data Dispatch Network: This block plays a core role in supporting dynamic spatial dataflow. In VersaCore, we propose an arbitrary dimensional (N-D) data dispatch network (DDN), which enables N-D multiplier array construction. As shown in Figure 13 (b), the N-D data dispatch network can be formulated as an N-D affine function, with the symbols from Table II. It conducts an extra data transformation from the sub-tensor that comes from DataMaestro. To support a Dsr multiplier array, a DDN with Dsr-dimensional affine data transformation should be instantiated, with the correct Bi for data element count to be dispatched and Si for data reuse patterns, for repeated (multicast), contiguous, or strided access of the sub-tensor elements. For example, for the spatial dataflow, the operands A and B both get an 8x8 sub-tensor from DataMaestro. The DDN for operand A will conduct a 3-D data transformation for the 8x8 sub-tensor, firstly contiguously accessing the first row, then repeating 8 times, then taking the next row in a strided way, and repeating again. The DDN for operand B will also conduct a 3-D data transformation for the 8x8 sub-tensor, firstly contiguously accessing the first column, then taking the next column in a strided way, then repeating this operation 8 times. Eventually, there are two 8x8x8 A and B tensors to be sent into the 3-D 8x8x8 spatial multiplier array. To support Ns spatial dataflows, Ns DDNs should be generated for each operand to accommodate the data dispatch pattern for different spatial dataflows. VersaCore can dynamically select the optimal DDN for each specific workload, enabling dynamic spatial dataflow.

3) Multiplier: The VersaCore has a multiplier library that contains multipliers with different data types, including various Int-Int, FP-Int, and FP-FP multiplications. Besides, it has a clean module interface and can be extended easily to employ a user-defined multiplier design and offer more workload customization.

4) Adder Tree: As shown in Figure 13 (c), the spatial adder is architected to a tree structure that has many stages of outputs to enable flexible spatial reduction, e.g., to support both the shallow and deep spatial reduction. It reuses the same input ports from the multiplier result for different stages, while dynamically directing the results of respective stages of the adder tree to output ports towards accumulators or DataMaestro according to the spatial reduction dimension configuration. This multi-stage result-directed design allows for the efficient reduction of multipliers' results across various dimensions (needs to be a power of 2) without the need for multiple dedicated adder trees, thus optimizing resource utilization. The maximal

number of adder tree stages is determined by the maximal spatial reduction dimension specified at generation time.

5) Accumulator: VersaCore enables temporal reduction (output stationarity) by putting an accumulation register at the adder tree output ports. Moreover, when temporal reduction is not required, such as in input/weight stationary scenarios, data gating can be applied to disable unnecessary accumulation and register access operations, thereby improving energy efficiency. The VersaCore array decouples the generation of its different modules, such as the data dispatch networks, multipliers, adders, adder trees, and accumulators, by modular design. These modules are reassembled in a hierarchical manner to formulate an integrated accelerator design, building the array in a bottom-up way. VersaCore adds multiplexers between these modules at the array level to dynamically construct the different spatial dataflows and utilize distinct data types. Besides, there is a hardware loop controller that manages the execution of the array operations and decoupled data handshaking, including clearing the accumulation register, giving correct data/configuration valid/ready signals, etc.

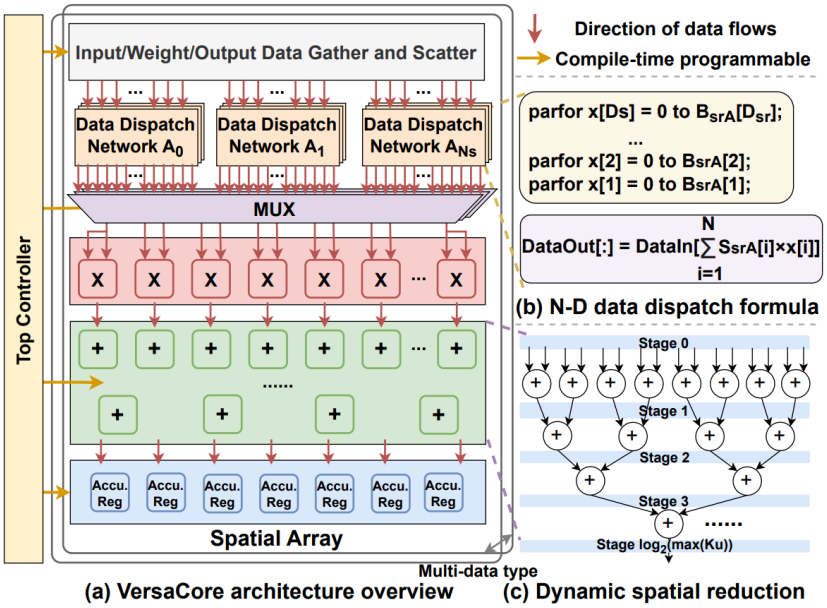


FIGURE 13: VERSACORE GENERATOR ARCHITECTURE

4.3. SNAX+VersaCore

The VersaCore naturally fits in the SNAX system for flexible dataflow accelerator generation. It also comes with a programming interface formally to the SNAX-MLIR compilers, as described in section 6.4. The overall system architecture is shown below. With SANX infrastructure, VersaCore's dynamic bandwidth requirement, posed by the dynamic spatial array, is met by the flexible data streamers. The SNAX+VersaCore's high flexibility enables an agile hardware generation flow that enables a dynamic dataflow based on the different computation characteristics, maximizing hardware utilization, and supporting full system generation, considering all the practical effects in actual workload deployment. Moreover, it aims to open up a vast design-space in the DNN acceleration system generation, facilitating benchmarking among different solutions and generating the optimal system output by architectural design-space exploration, either manually or by the DSE tools, with high design quality. This all strongly contributes to CONVOLVE's objective of enabling a drastic reductions in design time.

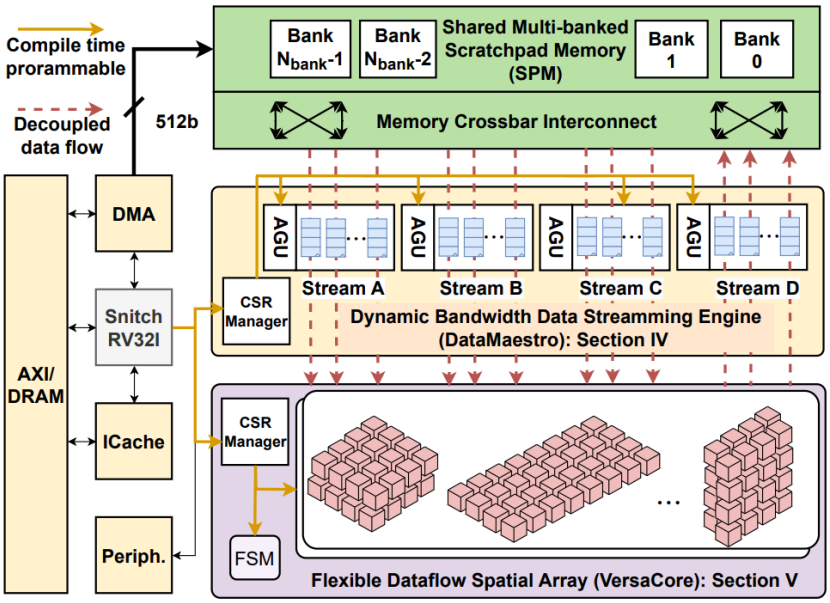


FIGURE 14: ARCHITECTURAL TEMPLATE OVERVIEW

5. Automated cluster and system integration, simulation and verification

5.1. Automatic Integration with Bender & Morty

5.1.1. Hardware Dependency Management

A key enabler of the CONVOLVE design flow at ETH Zurich was the adoption of Bender, an open-source tool for hardware dependency management. Bender provided a declarative and version-controlled mechanism to track, resolve, and assemble large sets of RTL sources originating from different intellectual property (IP) blocks. Its integration into the flow allowed us to manage dependencies across multiple design layers, from core microarchitectures, to clusters, and up to the SoC level, in a consistent and reproducible manner.

This modular approach facilitated parallel development: partner institutions could develop and iterate on their cluster designs independently, while ETH maintained a functional SoC top level using placeholder IPs. Once final RTL was delivered, these placeholders were seamlessly replaced with partner IPs. Thanks to Bender's "plug-in" integration model, the SoC template supported multiple configurations with minimal manual intervention. Building a complete system configuration became a single-step process, ensuring fast iteration and reducing integration overhead.

5.1.2. RTL Flattening and Version Control

Morty complemented Bender by addressing the challenges of integrating complex third-party IPs into the SoC. Morty automatically flattened hierarchical RTL designs into a single SystemVerilog file, which could then be frozen, version-controlled, and easily re-integrated.

This approach provided two major advantages:

- Traceability - Frozen versions ensured that synthesis, verification, and backend teams could work on a stable RTL snapshot while upstream development continued.

- Scalability – Large designs could be reliably managed. For example, the integrated KUL cluster in CONVOLVE generated a flattened file of approximately 8.3 MB (~200k lines of code), which was directly usable in downstream flows.

By combining Bender for dependency resolution and Morty for flattening, the toolflow enabled reproducible, automated, and version-safe integration of partner IPs into the SoC.

5.2. SoC Cluster Integration via Chimera

5.2.1. AXI Template and Adapter Support

The Chimera SoC template, developed at ETH Zurich, featured a well-defined AXI-based interconnect and integration model. Partner clusters could connect to the SoC with minimal manual wiring through a dedicated AXI adapter, designed specifically for CONVOLVE to standardize and simplify the integration process.

This standardized interface reduced integration complexity, eliminated ad-hoc wiring, and ensured compliance with the SoC's communication protocol. The approach was particularly beneficial when integrating heterogeneous partner clusters, each with distinct internal architectures, into a shared SoC environment.

5.2.2. Accelerated and Streamlined Integration of Partner Clusters

Using Chimera's AXI template and adapter, partner clusters could be integrated into the SoC with minimal modifications on either side. Once a cluster's AXI interface conformed to the \expected template, integration into the SoC required only the definition of address mappings and configuration parameters. This streamlined process enabled new cluster IPs to be swapped into the SoC quickly, supporting rapid design exploration without disrupting other system components.

As depicted in the Chimera SoC diagram, the combination of the Cheshire Host and the L2 Memory Island subsystem forms the backbone of the Chimera SoC template. The Cheshire Host, composed of a CVA6 RISC-V 32-bit core and a set of standard peripherals, acts as a general-purpose system host. Crucially, it provides the control and orchestration needed to offload computationally intensive workloads to the attached partner clusters, each of which can be specialized for different DNN applications.

The L2 Memory Island complements this design by offering shared memory space for inter-cluster communication and ensuring high-bandwidth access and data movement. This feature is essential to meet both the requirements of the individual clusters and the demanding memory throughput of modern DNN applications.

Thanks to the unified AXI interface, clusters designed by different partners can be plugged seamlessly into the template. This creates a heterogeneous and highly flexible cluster domain, where multiple specialized compute engines coexist within the same SoC. The resulting architecture demonstrates how Chimera can serve as a scalable, modular platform that accelerates integration while supporting a broad range of DNN workloads and research

directions.

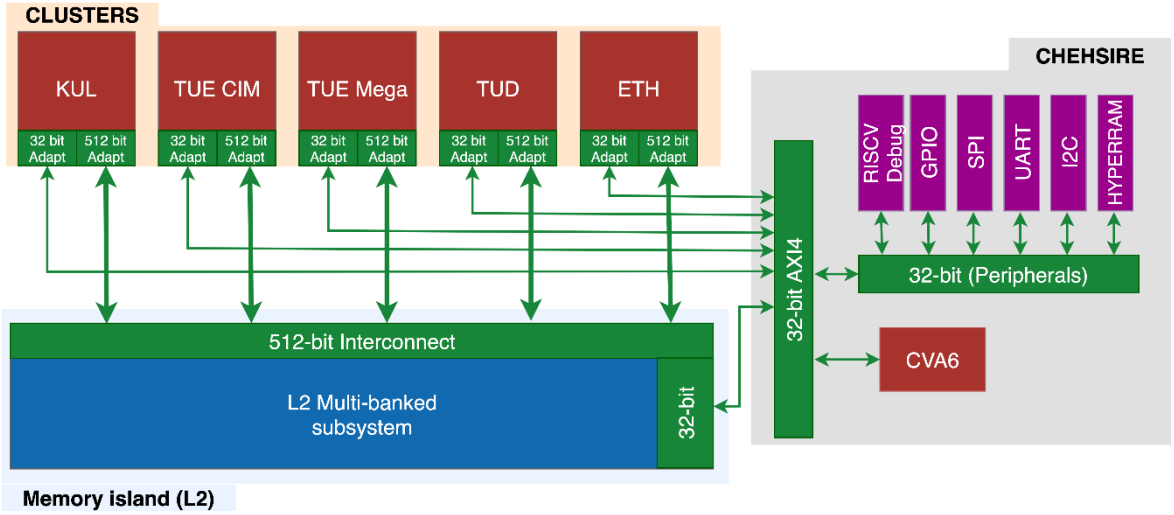


FIGURE 15: CHIMERA SOC TEMPLATE

5.3. Simulation and Testing Automation

5.3.1. Makefile-Based Build and Regression Flow

To ensure consistent build and test execution, a modular Makefile flow was developed. This automation wrapped around the Bender + Morty integration process, enabling:

- **Full hardware build automation**, from dependency resolution to flattened RTL generation.
- **Regression testing** via scripted simulation runs.
- **Configuration management** for building different SoC variants without manual source editing.

A key strength of this flow is its **cluster-agnostic design**. The Chimera platform can be simulated with different partner or internally developed cluster accelerators by simply changing the configuration, without requiring structural changes to the build system. This flexibility not only speeds up integration, but also enables efficient **design-space exploration**, allowing multiple architectural options to be evaluated under the same standardized simulation environment. As a result, new cluster designs can be plugged in, tested, and compared rapidly, both during CONVOLVE and in future research, significantly accelerating the overall SoC development process.

5.3.2. Makefile-Based Build and Regression Flow

To support hardware bring-up and addressability testing, a basic Hardware Abstraction Layer and software stack were provided alongside the SoC template. This included simple C-based functions to configure and interact with memory-mapped peripherals, as well as minimal runtime support for executing test programs on the integrated clusters.

The combination of HAL support and automated hardware build flows enabled the seamless execution of addressability and functional tests directly on simulated SoC instances, ensuring early detection of integration issues before physical prototyping or FPGA deployment. In addition, the automation framework substantially increased the efficiency of integrating new accelerators into the Chimera SoC template, thereby unlocking streamlined design-space exploration and enabling the evaluation of multiple heterogeneous compute engines within short iteration cycles.

From a quantitative perspective, the overall system complexity is substantial: approximately 20,000 lines of RTL were developed for the SoC top level and chip-level integration for tapeout, complemented by around 1,000 lines of regression and simulation automation. By contrast, integrating a new accelerator requires only limited and localized modifications. Specifically, adapting the system interface involves changes to the AXI adapter of approximately 440 lines, while the additional integration of the adapter and accelerator into the SoC amounts to around 230 lines. In total, the required modifications needed to integrate a new accelerator are on the order of 700 lines of code, representing a negligible fraction of the overall design complexity.

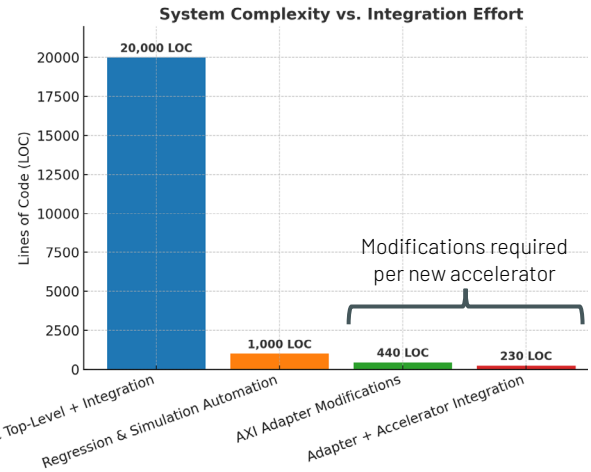


FIGURE 16: MODULAR AND TEMPLATE-BASED APPROACH ALLOWS TO ADD ACCELERATORS WITH MINIMAL RTL & CODE ADAPTATION.

This modular and template-based approach ensures that new accelerators exposing a compliant AXI port can be incorporated into the system with minimal engineering effort. The integration task is reduced from extensive manual wiring and restructuring to a concise and reproducible process, supported by automated build and regression flows. In this way, the Chimera SoC template provides a platform in which design-space exploration is accelerated by up to an order of magnitude compared to conventional SoC integration approaches, enabling faster prototyping, evaluation, and refinement of heterogeneous accelerators for edge AI workloads.

5.4. Summary of automated hardware generation and integration

Within CONVOLVE, ETH Zurich developed and deployed an integrated automation flow to accelerate SoC design, integration, and testing. The flow combined Bender for hardware dependency management, Morty for RTL flattening and version tracking, and the Chimera SoC template with AXI adapter support for low-effort partner IP integration. A modular Makefile-based build system further streamlined simulation and regression testing, complemented by a basic HAL and software stack for rapid bring-up and addressability validation. This toolchain enabled parallel development, reproducible builds, and fast design iterations, ensuring efficient integration of heterogeneous partner architectures into the final SoC.

6. Tools integration and co-operation: The Praxis toolflow

1. Praxis goals

Praxis represents a binding of the tools discussed above, towards a comprehensive scheduling-compilation-hardware exploration framework designed to address the challenge of huge Design Space Exploration (DSE) across the complete computing stack. The framework tackles the complex multi-layer optimization problem spanning from application requirements down to mapping, compilation, architecture, and circuit levels. More specifically:

1. Full-Stack DSE Integration: Praxis aims to bridge the gap between high-level AI workload descriptions and low-level hardware implementation by providing a unified framework that can simultaneously explore scheduling, compilation, and hardware generation aspects. This addresses the traditional problem of low agility in deep computing stack optimization, where different layers are typically optimized in isolation.
2. Pareto-Optimal Solution Generation: The framework aims to produce Pareto-optimal solutions that balance performance, power, and area constraints across the entire computing stack. This includes generating acceleration system RTL, RISC-V executable binaries, and optimal scheduling configurations simultaneously.
3. Hardware Template Flexibility: Praxis incorporates flexible hardware templates and generators, particularly focusing on SNAX+VersaCore architectures, allowing for rapid design space exploration across different hardware configurations.
4. Multi-Layer Optimization: The framework addresses optimization challenges spanning spatial and temporal mapping, memory space allocation, hardware cost model validation and calibration, functional verification, register and memory mapping, and tensor memory layout optimization.

Here, practically, we leverage Zigzag as the mapping and scheduling tool, the SNAX-VersaCore as the flexible hardware generation tool, Bender/Morty as the HW integration tool, and SNAX-MLIR as the compilation tool.

2. Linking ZigZag-Stream to VersaCore-SNAX

The integration between ZigZag-Stream and VersaCore-SNAX represents a critical component of the Praxis toolflow, enabling hardware-aware workload mapping and performance estimation. The key challenge here is the Zigzag cost model validation across different SNAX-VersaCore configurations and workload. Once this is finished, we can effectively leverage Zigzag to conduct comprehensive optimal workload mapping and hardware architectural search.

3. Linking VersaCore-SNAX to Compiler toolflow

The MLIR tool provides code generation based on the hardware architecture and the mapping, while ensuring the correct functionality, proper data movement, and data layout management, as well as proper synchronization. The connection between VersaCore-SNAX and the compiler toolflow is facilitated through SNAX-MLIR (Multi-Level Intermediate Representation), which provides a comprehensive compilation framework for SNAX accelerators. The compiler framework handles the translation from high-level AI workload descriptions to low-level hardware-specific code generation.

4. Linking ZigZag-Stream to Compiler toolflow

The direct integration between ZigZag-Stream and the compiler toolflow represents a novel approach to hardware-aware compilation and scheduling optimization. The integration includes mechanisms to constrain ZigZag mapping to even mapping patterns for tiling simplification, ensuring that the generated schedules are compatible with the compiler's code generation requirements. The MLIR gets optimal mapping from the ZigZag-Stream tool.

5. Early Use Cases and Validation

To realize a set of experiment, to assess the benefits of linking ZigZag, SNAX hardware generation and the MLIR compiler, we first benchmark the performance model of ZigZag, against RTL simulations. To this end, a SNAX cluster with an 8x8x8 GeMM cluster is created in hardware, and the same architecture is modelled in ZigZag.
Next, a wide variety of MatMul workloads are schedule on this processing core. On the ZigZag side, the execution time is estimated for each workloads using the ZigZag analytical models. In parallel, the same workload is simulated in RTL on the SNAX cluster, using Verilator simulation.
Figure 16 shows the benchmarking results of running 30 workloads in RTL and in ZigZag. As can be seen, the performance modelling matches well with less than 10% deviations.

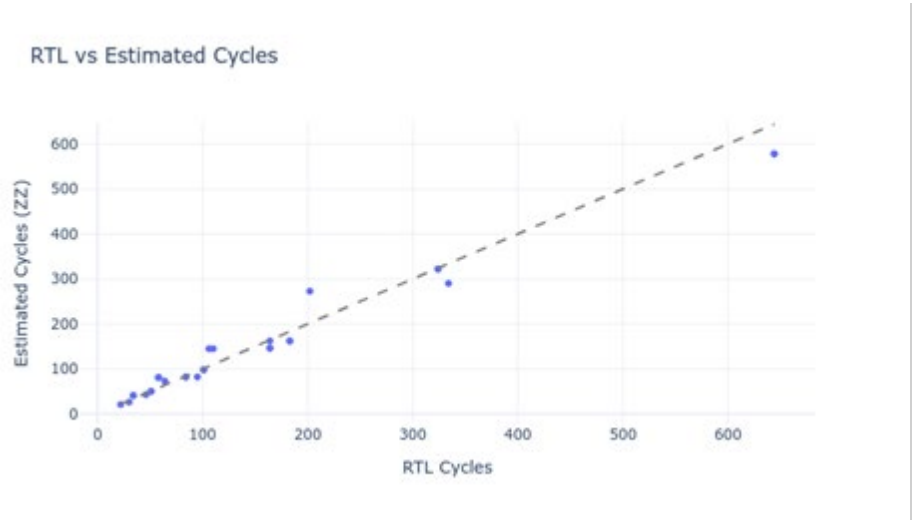


FIGURE 16: BENCHMARKING ZIGZAG LATENCY ESTIMATIONS VERSUS RTL SIMULATION

The next step, is to use ZigZag to optimize tiling for SNAX execution, and let the compiler translate this tiling scheme into a binary executable. It is hereby important to realize that the scratchpad memory for weights, inputs and outputs is the same shared memory in SNAX. The tool hence has to optimize the tiling, ensuring the sum of all memory contributions fits in the SNAX cluster memory.
Figure 17 shows the resulting memory footprint of tiled execution of ResNet50 on SNAX, showing a fit into the available 512kB of memory. In case the different inputs, output and weight memories would instead not be shared, but private, almost 2x this amount of memory would have been needed (Figure 17).

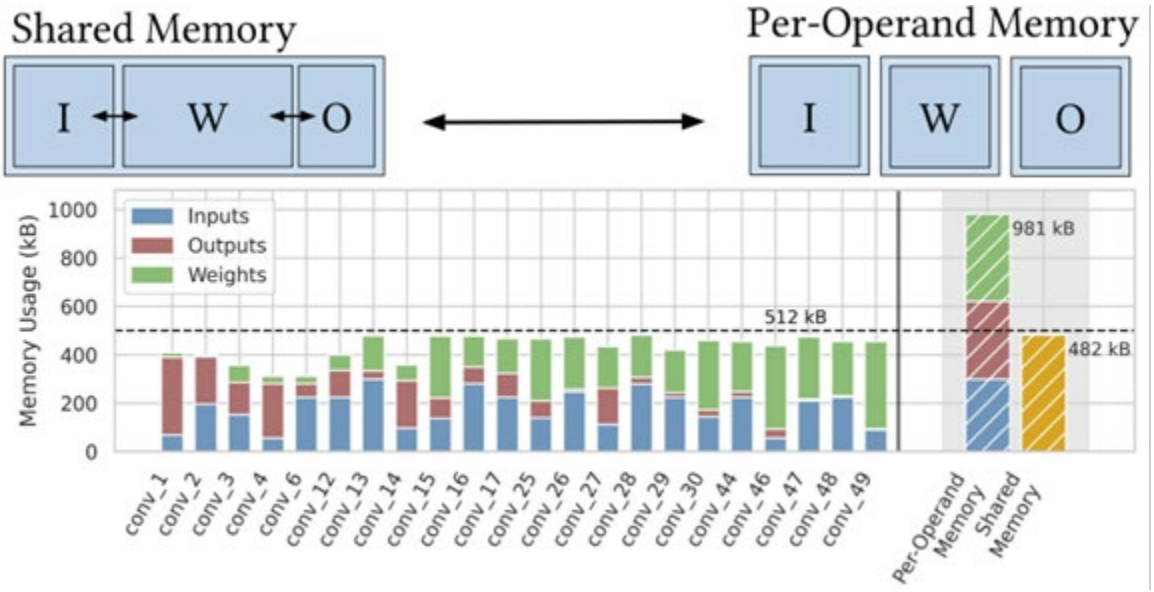


FIGURE 17: THE SHARED MEMORY ALLOWS FOR DYNAMIC TILE SIZES TO MINIMIZE DRAM TRANSFERS (FOR RESNET-50)

Finally, bringing these techniques together with other compiler optimization techniques, such as double buffering for accelerator configuration and layout transformations allows SNAX to operate very close to its theoretical roofline performance, hence demonstrating a near perfect utilization (Figure 18).

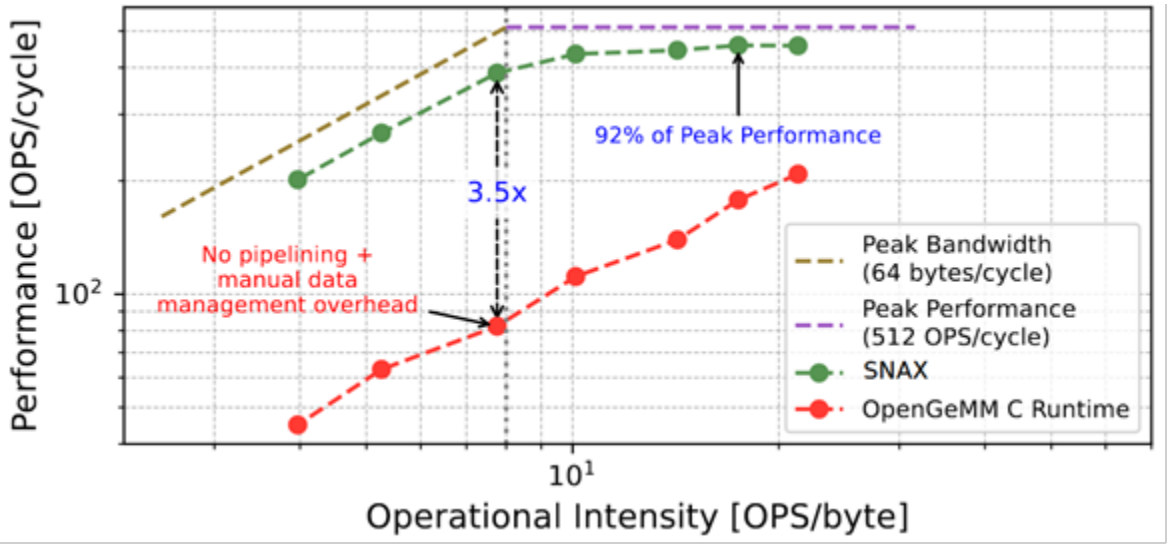


FIGURE 18: THE COMPILER OPTIMIZES TILED MATMUL WITH DOUBLE BUFFERING + STATIC LAYOUT TRANSFORMATIONS + CONFIG OVERHEAD ELIMINATION ALLOWS TO ACHIEVE PERFORMANCES VERY CLOSE TO THE ROOFLINE.

6. Outlook and Opportunities

The Praxis framework is in constant evolution. It presents several significant opportunities for future development and broader adoption. More specifically, it enables:

1. Enhanced DSE Capabilities: Future development will focus on expanding the design space exploration capabilities to include more sophisticated search engines beyond random and exhaustive approaches. This can include algorithm-based optimization techniques and machine learning-guided exploration strategies to efficiently navigate the vast design space.
2. Broader Accelerator/Workload Support: The framework architecture supports extension to additional accelerator types beyond single GEMM operations. This

includes specialized accelerators for different AI workload patterns, custom compute units, and emerging AI algorithms such as LLMs and MOE.

3. Academic and Research Impact: The comprehensive nature of the Praxis framework positions it as a valuable research platform for exploring future AI accelerator architectures. The framework's ability to co-optimize across multiple layers of the computing stack enables novel research directions in hardware-software co-design. The open-source nature of key components (ZigZag, SNAX-MLIR, SNAX-VersaCore) provides clear pathways for fast idea prototyping, and the framework's ability to generate synthesizable RTL and executable binaries makes it attractive for agile AI accelerator development.

7. Conclusions and Outlook

The CONVOLVE project has developed and integrated toolflows that enable efficient system-on-chip (SoC) generation and design-space exploration (DSE) for edge AI hardware, through the integration of modelling frameworks, automated hardware generation tools, and system-level validation infrastructure.

The toolflow brings together key components: Stream, ZigZag, VersaCore, SNAX, Bender and Morty, to facilitate rapid exploration of design-spaces and generate optimal SoC architectures. The integrated scheduling and mapping frameworks enable real-time evaluation of workload-specific accelerators and the application of fault tolerance models for safety-critical domains. The toolchain has been validated through the design and tape-out of the Chimera SoC, demonstrating its potential to streamline the design and prototyping of modular SoCs.

7.1. Summary of Toolflow Integration Achievements

The integration of multiple tool components within the CONVOLVE project has yielded an automated, scalable, and flexible design flow. Key achievements include:

- The extension of the Stream and ZigZag frameworks for efficient workload mapping, scheduling, and memory management, incorporating SDF-based models and MILP optimization techniques. GIT: https://github.com/KULeuven-MICAS/zigzag and https://github.com/KULeuven-MICAS/zigzag
- The integration of fault tolerance modelling into the DSE flow, allowing the exploration of selective protection strategies (e.g., DMR and TMR) for enhancing reliability without incurring prohibitive overheads.
- The development of the SNAX+VersaCore platform, enabling rapid generation of flexible AI accelerators based on workload-specific dataflows, improving both performance and energy efficiency. GIT: https://github.com/kuleuven-micas/snax_cluster
- System-level integration and validation using the Chimera templates, Bender, and Morty, enabling automated integration of partner IPs and SoC prototypes, with streamlined simulation and regression testing flows. GIT: https://github.com/pulp-platform/chimera
- The Praxis framework for full-stack optimization, linking high-level workload modelling to low-level hardware generation and compiler flows, ensuring efficient and scalable SoC design.

These advancements collectively contribute to reducing time-to-market for AI accelerator-based SoCs, while improving their performance and energy efficiency. The toolchain provides a robust foundation for future work in edge AI hardware design and the scaling of AI systems.

7.2. Lessons Learned

Several key lessons emerged during the development of the CONVOLVE toolchain, highlighting the importance of agility, modularity, and robust validation in building complex design flows for SoC-based accelerators:

- Flexibility in architecture is critical for accommodating a broad range of workloads. The VersaCore-SNAX platform demonstrated that customizable, workload-specific dataflows lead to more efficient hardware while maintaining flexibility for future optimizations.
- Toolchain integration complexity: Integrating diverse tools like Stream, ZigZag, VersaCore, Bender and Morty into a seamless flow was a complex task, but it underscored the importance of modular design. Building tools that can easily integrate with one another helps mitigate risks and accelerates development.
- Scalability of simulation and testing: Achieving rapid prototyping and validation of multi-core SoCs requires careful planning of simulation frameworks. The Chimera-template based integration provided a scalable approach to multi-cluster simulation.
- Energy-efficiency vs. complexity trade-offs: While increasing the robustness of AI accelerators (e.g., through redundancy) improves reliability, it must be carefully balanced with the overhead costs in terms of latency and energy consumption. The integration of fault tolerance models into the DSE allowed for systematic trade-off analysis.

7.3. Future Work and Recommendations

Future work will focus on further enhancing the toolflow to ensure its continued relevance in edge AI hardware design:

- Expanding support for new accelerator types: Currently, the toolchain primarily supports DNN-like AI dataflow accelerators. Extending the framework to support new accelerator paradigms, such as hardware for LLMs, Neuro-symbolic AI, and Splaing based algorithms, will broaden its applicability.
- Further integration with machine learning models: Integrating the toolchain with machine learning-based optimization techniques, such as reinforcement learning for DSE, could enable more efficient exploration of design-spaces.
- Automating multi-objective optimization: Future versions of the Praxis framework should integrate advanced multi-objective optimization algorithms to handle more complex design-space exploration, helping to automatically select the optimal configuration based on multiple, competing objectives (e.g., performance, energy, cost).
- Fault tolerance for deep learning models: Building upon the initial fault tolerance models, future work should focus on incorporating soft error resilience at a finer granularity, considering the impact of errors on model accuracy in safety-critical applications.
- Open-source adoption and community involvement: To facilitate rapid adoption and improve the toolchain, all key components are already released in the open-source (e.g., ZigZag, SNAX-MLIR, VersaCore). Yet, documentations and tutorials can be improved further. This would enable academic and industrial researchers to extend the toolchain for novel AI applications and ensure continuous feedback from the community.

8. References

[R1] G. Li, S. K. S. Hari, M. Sullivan, T. Tsai, K. Pattabiraman, J. Emer, and S. W. Keckler, "Understanding error propagation in deep learning neural network (dnn) accelerators and applications," in Proceedings of the international conference for high performance computing, networking, storage and analysis, 2017.

[R2] B. Reagen, P. Whatmough, R. Adolf, S. Rama, H. Lee, S. K. Lee, J. M. Hern´andez-Lobato, G.-Y. Wei, and D. Brooks, "Minerva: enabling low-power, highly-accurate deep neural network accelerators," in Proceedings of the 43rd International Symposium on Computer Architecture, ser. ISCA '16. IEEE Press.

[R3] H. D. Dixit, S. Pendharkar, M. Beadon, C. Mason, T. Chakravarthy, B. Muthiah, and S. Sankar, "Silent data corruptions at scale," arXiv preprint arXiv:2102.11245, 2021.

[R4] A. Ruospo, G. Gavarini, I. Bragaglia, M. Traiola, A. Bosio, and E. Sanchez, "Selective hardening of critical neurons in deep neural networks," in 2022 25th International Symposium on Design and Diagnostics of Electronic Circuits and Systems (DDECS). IEEE.

[R5] I. Baek, W. Chen, Z. Zhu, S. Samii, and R. Rajkumar, "Ft-deepnets: Fault-tolerant convolutional neural networks with kernel-based duplication," in Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision, 2022.

[R6] A. Mahmoud, S. K. S. Hari, C. W. Fletcher, S. V. Adve, C. Sakr, N. R.Shanbhag, P. Molchanov, M. B. Sullivan, T. Tsai, and S. W. Keckler, "Optimizing selective protection for cnn resilience." in ISSRE, 2021.

[R7] M. H. Ahmadilivani, S. Mousavi, J. Raik, M. Daneshtalab, and M. Jenihhin, "Cost-effective fault tolerance for cnns using parameter vulnerability based hardening and pruning," in 2024 IEEE 30th International Symposium on On-Line Testing and Robust System Design (IOLTS).

[R8] R. Baumann, "Radiation-induced soft errors in advanced semiconductor technologies," IEEE Transactions on Device and Materials Reliability, 2005.

[R9] K. Ueyoshi, I. A. Papistas, P. Houshmand, G. M. Sarda, V. Jain, M. Shi, Q. Zheng, S. Giraldo, P. Vrancx, J. Doevenspeck, D. Bhattacharjee, S. Cosemans, A. Mallik, P. Debacker, D. Verkest, and M. Verhelst, "Diana: An end-to-end energy-efficient digital and analog hybrid neural
network soc," in 2022 IEEE International Solid-State Circuits Conference (ISSCC).

[R10] H. Jia, M. Ozatay, Y. Tang, H. Valavi, R. Pathak, J. Lee, and N. Verma, "Scalable and programmable neural network inference accelerator based on in-memory computing," IEEE Journal of Solid-State Circuits, 2022.