Progress Report: "PACE: Next-Generation IoT Edge Computing through Efficient Software Programmable Accelerators"

We report the technical achievements and the progress made in the project till March 2025.

The overall objective of the PACE project is to deliver highly efficient software programmable accelerators based on Coarse-Grained Reconfigurable Architecture (CGRA) that can offer near ASIC-like compute efficiency, i.e., performance-per-watt while allowing full software programmability. The concrete goal is to bring server-class processing capability to the resource-scarce edge devices at ultra-low power through a synergistic cross-layer approach. We expect to reach up to 50X improvement in compute efficiency compared to the general-purpose processors in state-of-the-art edge devices. We will realize this breakthrough in powerful edge computing by adopting an end-to-end approach with leading-edge innovations spanning applications, algorithms, compiler, architecture, and circuits.

A) Progress towards PACE 1.0 Accelerator Chip

The key deliverable of the project is the PACE 1.0 accelerator chip with the following specifications: 500 GOPS/W and 1hr battery life at weight of 1kg by the mid-term mark of the project (in Sep 2023). We have successfully achieved this goal.

S/N	Milestones	Status
1	PACE 1.0: CGRA accelerator chip: 500 GOPS/W peak performance with bit-width configurable ALU and buffer-less router	**Completed** • PACE 1.0 chip sent for fabrication in Dec 2023. Expected fab/out in Apr. • The PACE accelerator chip that was sent out for fabrication in Dec 2022 has been tested and measured. The PACE CGRA can run at a max frequency of 105 MHz while only consuming 43mW at 1V supply (100% PE utilization), achieving a maximum energy efficiency of 360 GOPS/W at 0.6V and 21MHz.
2	PACE 2.0 CGRA accelerator chip: 2000 GOPS/W with Near-memory compute macro and Fine-grained power management	**On-going** • We have moved to 12nm for better efficiency • We have achieved ~2TOPS/W in simulations by employing SIMD architecture with support for Floating point operations for quantized AI workloads

We have now successfully demonstrated execution of complete application on PACE 1.0 chip [28, 29].

Currently, our design achieved a simulated peak energy efficiency of *142 GOPS/W, 462 GOPS/W and 568 GOPS/W at 0.9 V, 0.5 V and 0.45V, respectively*. With this figure of merit, we showed very competitive performance, although not the best when compared to the state of the arts in Figure 1. However, note that ISSCC'19 and SSCL'20 from KU Leuven use 22nm FDX and 28nm CMOS processes, respectively while we are using 40nm. *If process normalization is considered, PACE 1.0 offers a peak energy efficiency of 673 GOPS/W and ~1090 GOPS/W when normalize to 28nm and 22nm processes, respectively*.

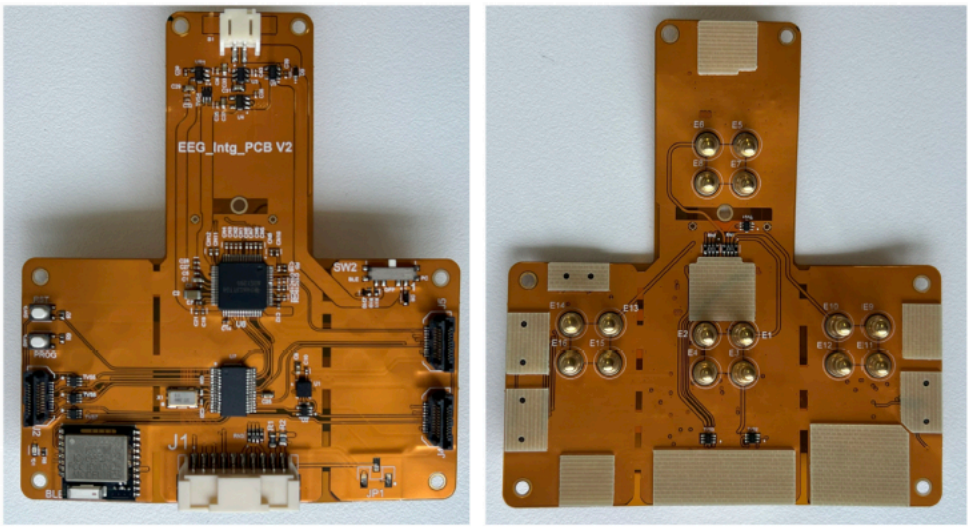
Parameters	TVLSI'18 Fudan Uni.	ISSCC'19 KU Leuven	ASSCC'19 NUS		JSSC'20 Tsinghua	SSCL'20 KU Leuven	PACE
Tech. (nm)	55	22 FDX	40		28	28	40
Supply voltage (V)	1.2	0.8	1.1		0.9	0.4-1	1
No. of Pes	25 + 5	15PE x 22ALU	16		64 (4x4x4)	120	64 PEs x 1 ALU
Area (mm^2)	5.19	4.9	2.87		4.8	3.9	3.02[1]
Freq. (MHz)	450	5-220	853	753	800	>300	100
Data bit-width	16	16/32	32		1/2/8/12/22/32	16/20/24	16
No. of Instructions	--	--	--		26	-	17
Power per CGRA core (mW)	~17.4	0.67@30MHz	2.88@853MHz		7.5	0.175@15MHz	0.7@0.9V
OPS per core per cycle	6.8	22	1		1	1	1
Core power efficiency (GOPS/W)	176	985	296		106	85	142
Norm. core power efficiency[2] (GOPS/W)	679	608	604		106	85	142
Memory (KB) (% Power)	54 KB (23.6)	690 KB (0-8.5)	7 KB (3 + 4) (23.6)		320 KB (8.7)	234 KB (0-3.4)	80 KB[3] (29.3%)
Power (mW)	841	10@30MHz	72	242	528	21 @15 MHz	45
Peak performance (GOPS)	77.4	145@220MHz	6.48	5.3	102.4	172.6	6.4
Applications	I. Processing ML	I. Processing	IoT		Massive MIMO	Various	Various
Efficiency (GOPS/W)	50.78	978@36MHz, 0.48V	90	22	196 (102.4)	585@15MHz, 0.6V (103 tap FIR)	142
Norm. Efficiency[2] (GOPS/W)	192	603	183	45	102.4	585	142@0.9V 462@0.5V

[1]Layout area of PACE, [2]Normalized to 40nm, [3] 64KB data memory + 16KB config memory

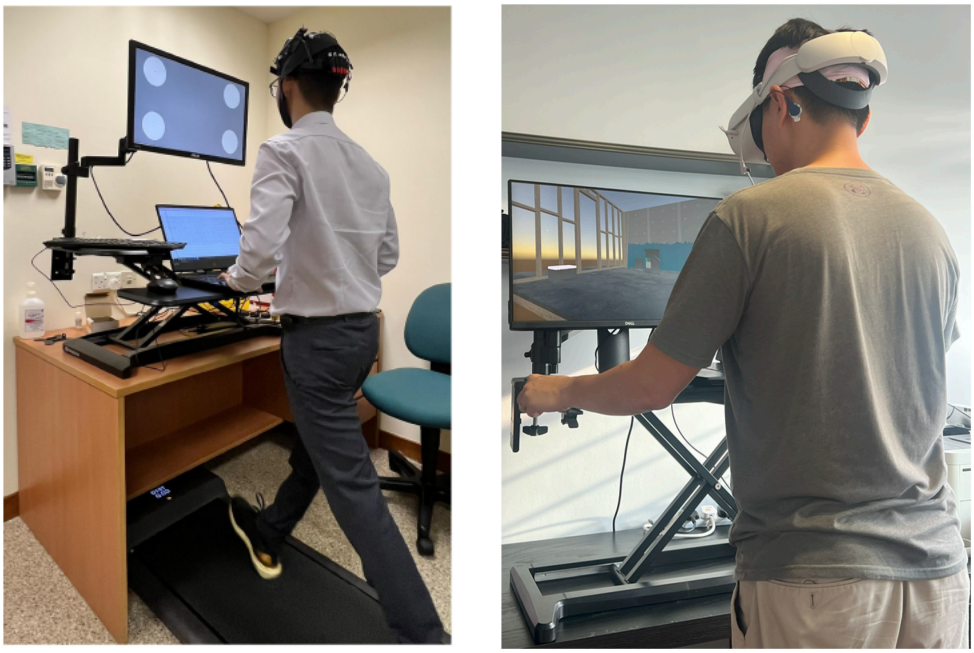
Figure. 1 Performnace comparison table against state-of-the-art accelerator chips.

We have leveraged the PACE 1.0 accelerator chip to enhance the performance of EEG devices. The PACE 1.0 chip aligns well with the demands of wearable EEG systems, providing the necessary computational power while maintaining portability and energy efficiency. WalkingWizard, a truly wearable EEG headset designed for everyday use, integrates dry electrodes with flexible PCB assembly to achieve accurate EEG sensing even under motion. By incorporating the PACE 1.0 chip, we aim to improve the real-time processing capabilities of WalkingWizard, enhancing its performance in capturing and analyzing EEG signals during various activities, such as walking or engaging in VR environments. This integration will facilitate the development of advanced applications like real-time emotion monitoring and cognitive state assessment, ultimately pushing the boundaries of mobile brain-computer interfaces and making EEG technology more accessible and practical for everyday use. We have mapped the EEG application on the PACE 1.0 accelerator chip and designed the new PCB system to integrate the chip in the system.

WalkingWizard Cap



Flexible PCB Assembly of WalkingWizard



WalkingWizard Usage

Details of our progress are provided below.

A.1) PACE 1.0:

In order to develop the working prototype including hardware and software for the PACE 1.0 accelerator chip, we have taped out our chip. The design has been taped-out by IME in UMC40nm CMOS. The block diagram of PACE accelerator is given in Figure 2. It consists of an 8x8 matrix of computation cores (i.e. PE) in the centre and 8 data memory modules on the left and right sides of the PE matrix, which are implemented using dual-ports RAMs. Each PE supports 17 instructions (Fig. 2(b)), which are co-optimized and selected via analysing the major kenels required for edge computing wordloads.

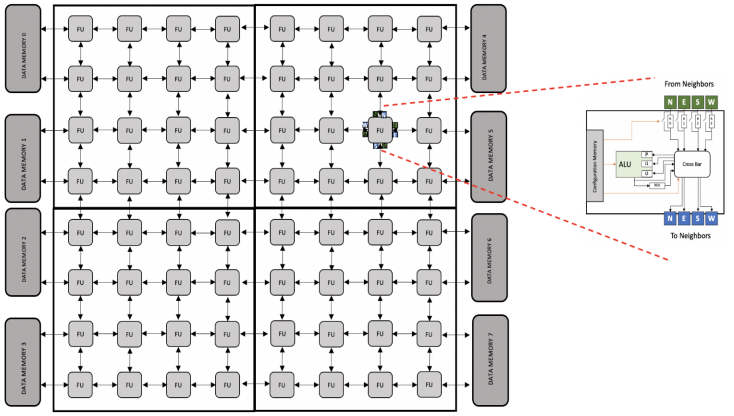


Table 3.3: Operations of the ALU module

Operation	Opcode	Description
NOP	5'b00000	No operation
ADD	5'b00001	Unsigned addition
SUB	5'b00010	Unsigned subtraction
MULT	5'b00011	Signed multiplication
LS	5'b01000	Left shifter
RS	5'b01001	Right shifter
ARS	5'b01010	Algorithm shifter
AND	5'b01011	Bitwise and
OR	5'b01100	Bitwise or
XOR	5'b01101	Bitwise xor
SEL	5'b10000	Operand selection
CMERGE	5'b10001	Operand selection
CMP	5'b10010	Equal-to comparison
CLT	5'b10011	Less-than comparison
BR	5'b10100	Multiple bitwise or
CGT	5'b10101	Greater-than comparison
MOVC	5'b11111	Operand move

Figure. 2 (a) Block diagram of PACE accelerator (b) supported Instructions

As also shown in Figure 2, each tile (i.e., PE) consists of (a) a configuration memory for dynamically setting the behavior and connectivity of the tile; (b) An ALU for performing logic computation such as add, shift and multiplication; (c) Input registers R0-R4 and input/output crossbar to handle the data flow between different tiles. As the key requirements of hardware accelerator for Internet-of-Things (IoT) edge computing applications, our focus is to improve the energy-efficiency of the PE. With an improved arithmetic logic unit (ALU) and memory integration, the PE power consumption is reduced up to 19.06% to ~700 µW at 0.9V, 100 MHz [6].

CGRA workload differs depending on the applications and kernels. We explore this characteristic and embed No-Operation (NOP) instruction with hardware infrastructure shown in Figure 3 so that individual tile can be clock gated by the complier. With the NOP and clock-gating optimization, the system power consumption in idle mode is reduced by 24%. Fig. 4 shows the power consumption of the ALU when performing difference instructions.

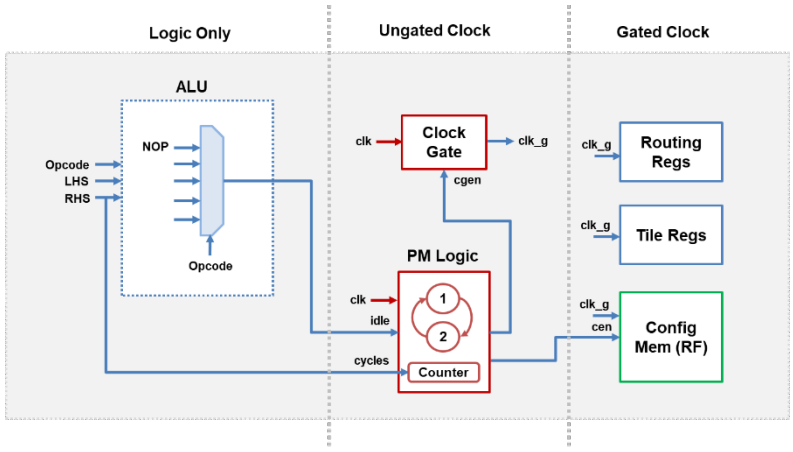


Figure 3. PACE NOP state microarch and optimization results

Operation	Power (uW)	Operation	Power (uW)
ADD	54	XOR	38
SUB	54	SEL	39
MULT	123	MERGE	38
LS	49	CMP	34
RS	49	CLT	42
ARS	49	BR	40
AND	39	CGT	42
OR	39	MOVC	38
Random	75		

Figure 4. Power for each ALU operation (100MHz, 0.9V)

A.2) RISC-V Based SoC

Although our focus is to develop highly energy-efficient PACE accelerator, our analysis showed that it is extremely inefficient to operation PACE as a stand-alone chip with an external microcontroller. Not only this greatly reduces the data transfer bandwidth (and thus performance) but it also incurs additional power consumption due to data transfer from chip to chip.

Thus, we decided to integrate PACE together with a RISC-V processor (i.e., Rocket CPU) in SoC, as shown in Figure 5. The top architecture is also shown in Figure 5. Within this SoC chip, PACE accelerator will work as one peripheral of the RISC-V core.

The Rocket CPU core is based on the RISC-V Instruction Set Architecture (ISA). It is a 5-stage in-order, generatable, and scalar core that implements the RV32G/RV64G ISAs. It has an MMU that supports page-based virtual memory, a non-blocking data cache, and a front-end with branch prediction. Branch prediction is configurable and provided by a branch target buffer (BTB), branch history table (BHT), and a return address stack (RAS). For floating-point, Rocket makes use of Berkeley's Chisel implementations of floating-point units. Rocket also supports the RISC-V machine, supervisor, and user privilege levels. Several parameters are exposed, including the optional support of some ISA extensions (M, A, F, D), the number of floating-point pipeline stages, and the cache and TLB sizes.

The Rocket CPU core can also be thought of as a library of processor components. Several modules originally designed for Rocket are re-used by other designs, including the functional units, caches, TLBs, the page table walker, and the privileged architecture implementation (i.e., the control and status register file). Figure 5 shows the internal components of a Rocket Chip instance, which includes the Rocket Core and various other subsystems.

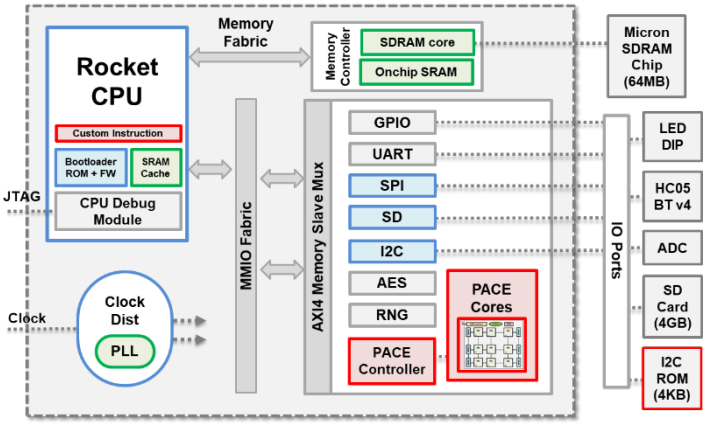


Figure 5. PACE-V SoC with Rocket CPU

The Rocket core is heavily customized to make it suitable for integration within the PACE-V SoC ecosystem and is extensively validated. Rather than being a single instance of an SoC design, the Rocket core generator can produce many design instances from a single high-level source. It produces design instances consisting of synthesizable RTL, and multiple functional silicon prototypes have been manufactured. Extensive parameterization makes it extensible, enabling easy customization for a particular application. Users can customize and generate SoCs ranging in size from embedded microcontrollers to multi-core server chips by just changing the related configuration files.

The BootROM firmware for the Rocket core is customized for the SoC:

- Fully written in RISC-V assembly for reduced hardware footprint (approximately 200 bytes).
- Fetches executable binary from external I2C EEPROM chip.
- Includes a prebuilt bootup delay time (50ms) for system stability.
- BootROM firmware binary header contains various hardcoded settings, e.g., Memory size (8KB), Speed (400KHz), Vendor ID, address.
- The boot flow is shown in Figure 6, whereby the first firmware is fetched from the external I2C EEPROM chip and placed in the system memory.

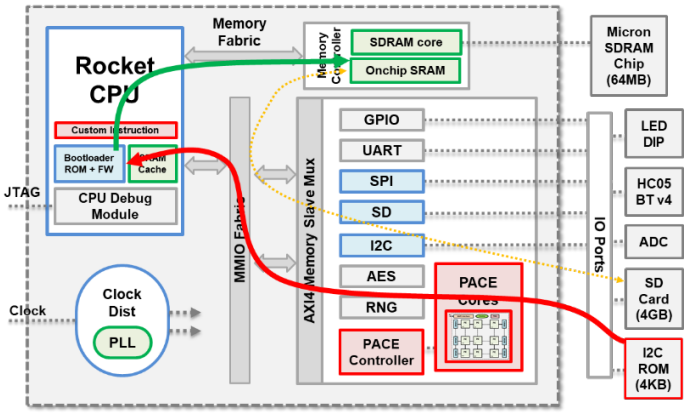
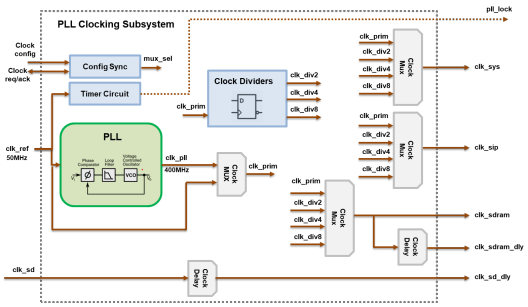


Figure 6. PACE-V SoC bootup flow

A.3) Clock root for RSIC-V PACE-SoC

For PACE v1.0 tape out, a clock root is implemented to supply clock source to the entire design. As shown in Figure 7, clock signals are provided to the RISC-V based microcontroller and the PACE CGRA. The maximum frequency is 400MHz. Internally, 3 levels of division, i.e., divide-by-2, 4, and 8, are used to reduce clock frequency down to 50MHz. Figure 8 shows that layout of the clock root. Post-layout simulation is performed to verify the outputs of the clock root.



Clock	Frequency	Type	Purpose
clk_ref	50 MHz	Input	Reference Clock for PLL
clk_pll	400 MHz	Internal	PLL HIP output clock
clk_div*	50-400MHz	Internal	Clock divider array
clk_sys	50-400 MHz	Output	System clock
clk_sip	50-400 MHz	Output	Sub IP clock
clk_sdram	50-400 MHz	Output	SDRAM clock
clk_sdram_dly	50-400 MHz	Output	SDRAM clock with configurable delay
clk_sd_dly	50-400 MHz	Output	SD card clock with configurable delay
clk_pll_view	50-400 MHz	Output	Debug pin, follow clk_sys

Figure 7. Schematic and specifications of the clock root for PACE v1.0

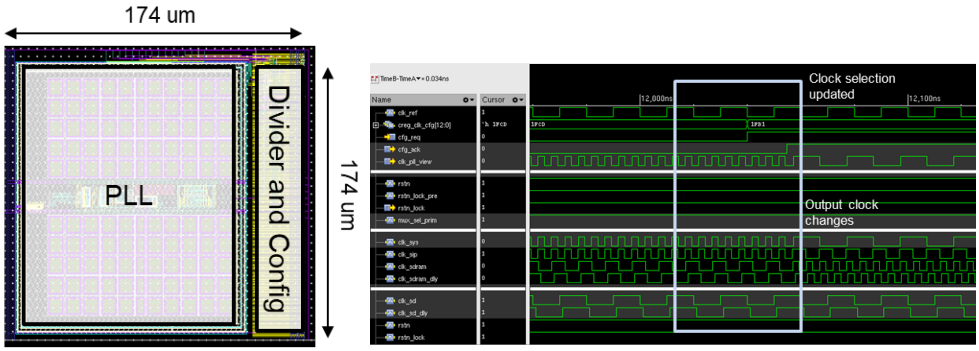
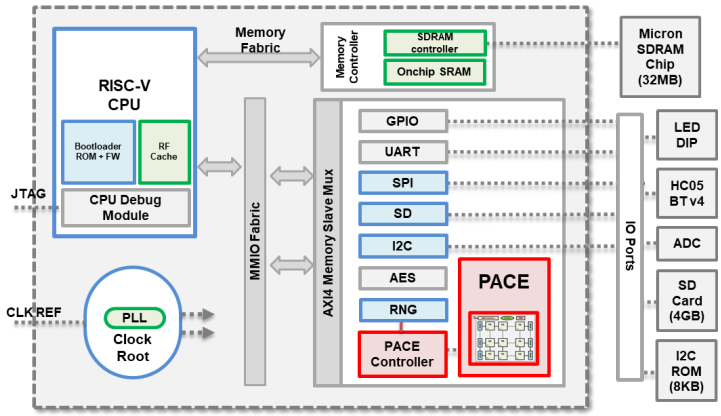


Figure 8. Layout of the clock root for PACE v1.0 and a snapshot of its postlayout simulation.

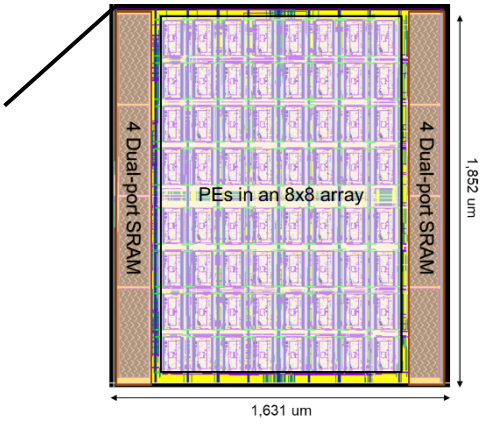
A.4) Physical implementation and performance analysis

Final system diagram and specifications of the RISC-V PACE SoC design are shown in Figure 9 while chip layout and bonding diagram are shown in Figure 10 and 11, respectively. We are finalizing the PCB layout and fabrication; the initial version is given in Figure 12.



PACE SoC Specifications		
Core and peripheral	Tech. Node	UMC 40nm ULP
	Supply Voltage	0.9V/3.3V
	Processor	RISC-V CPU (5 stage Rocket Core)
	Primary memory	Micron MT48LC64M4A2 SDRAM (32MB)
	Supported Comm.	GPIO×16, SD card, SPI×4 & UART
	Debug	JTAG
	On-chip SRAM	512 KB
	System clock freq.	200/100 MHz
	Core power	15mW @ 0.9V
	iCache/dCache	16KB / 16KB
	Custom bootloader	Yes, external EEPROM
	Operating Temp.	0°C - 85°C
PACE HW	Number of PEs	8x8
	PACE clock freq.	100 MHz
	Data memory	8 x 8KB dual-port SRAM
	#instructions	17, including NoP
	Config memory	64 x 0.25KB single-port SRAM
	PACE power	45 mW @ 0.9V
	PACE Peak Efficiency	142 GOPS/W
	Interface	AXI MIMO (32 bits)

Figure 9. PACE and RISC-V core integration



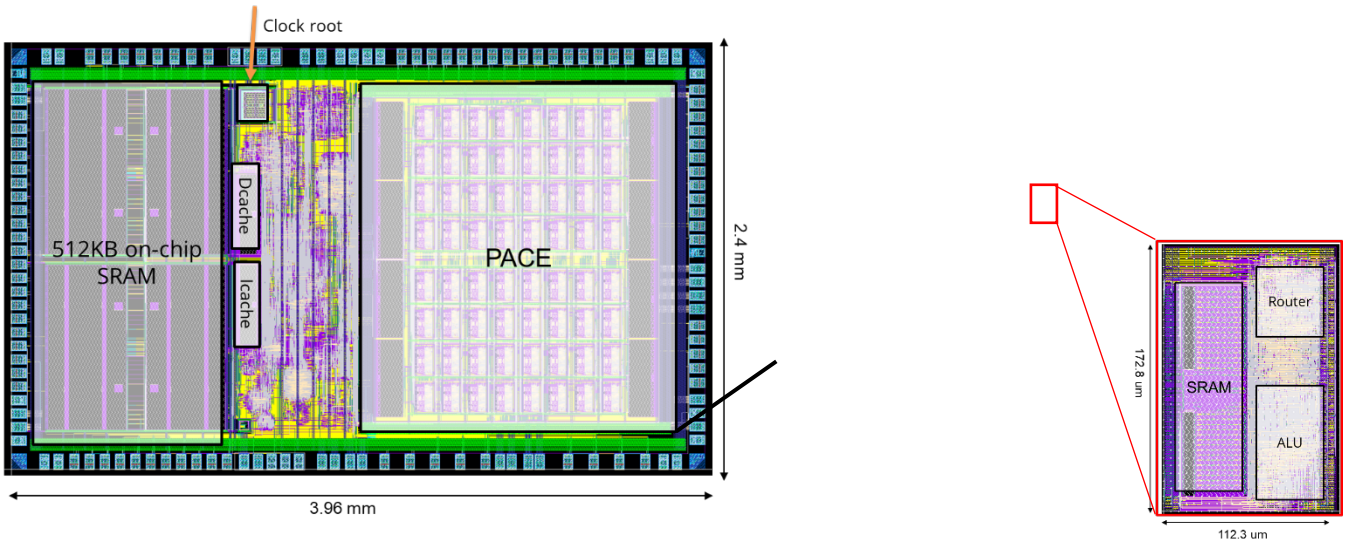


Figure 10. Chip layout for PACE v1.0

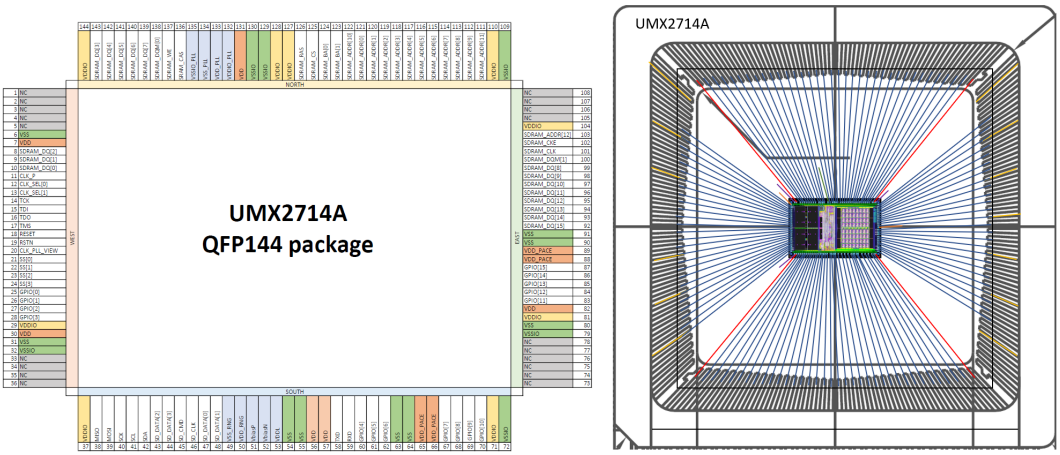


Figure 11. Pin out and wire bonding diagram for PACE v1.0

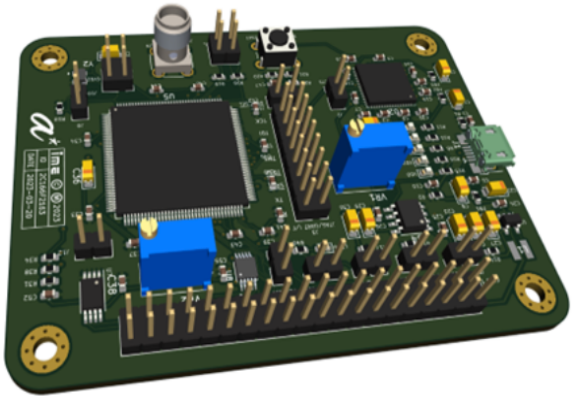
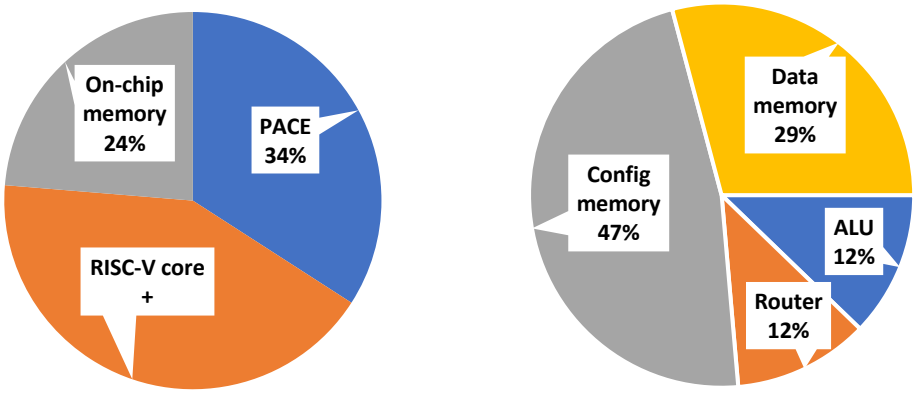


Figure 12. PCB design for PACE 1.0

Fig. 13 shows the layout area breakdown of the whole SoC. PACE takes up about 30% of the die size (4mm by 2.4mm). PACE consists of 8 pieces of data memory of 8K and 64 PEs. Each PE consists of configuration memory of 0.25KB, router and arithmetic logic unit (ALU). The 64 PEs take up 71% of the area of PACE. Figure 14 shows the power breakdown when the PACE is simulated with micro-speech kernel. The PEs takes up most of the power, i.e., 97%, during the processing.

Among the components in PEs, most of the power is consumed by the configuration memory, followed by the logics in the PEs which coordinate the operation based on the instructions.



(a) (b)

Figure 13. Area breakdown for (a) RSIC-V PACE SoC and (b) PACE accelerator.

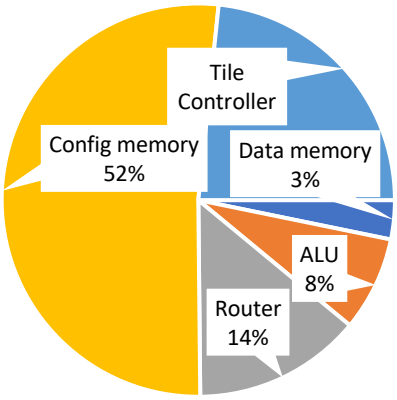


Figure 14. Simulation power breakdown when PACE runs microspeech kernel at 100MHz, 0.9V

A.5) Emulation of PACE architecture on FPGA

We also migrated the PACE RTL design to the Digilent Genesys 2 FPGA (Xilinx Kintex-7™ FPGA (XC7K325T-2FFG900C)) board for verification and application demonstration purposes. With the clock generator being mapped to MMCM module and SRAM being mapped to the block ram module, the emulation of the PACE design is successful on the evaluation board (Figure 15). As of March 2023, we have successfully verified two applications (Array Add and MicroSpeech) running on FPGA-based PACE design. Two FPGA boards (one in IME and one in NUS) are currently deployed to enable our team members to develop different applications on this platform.

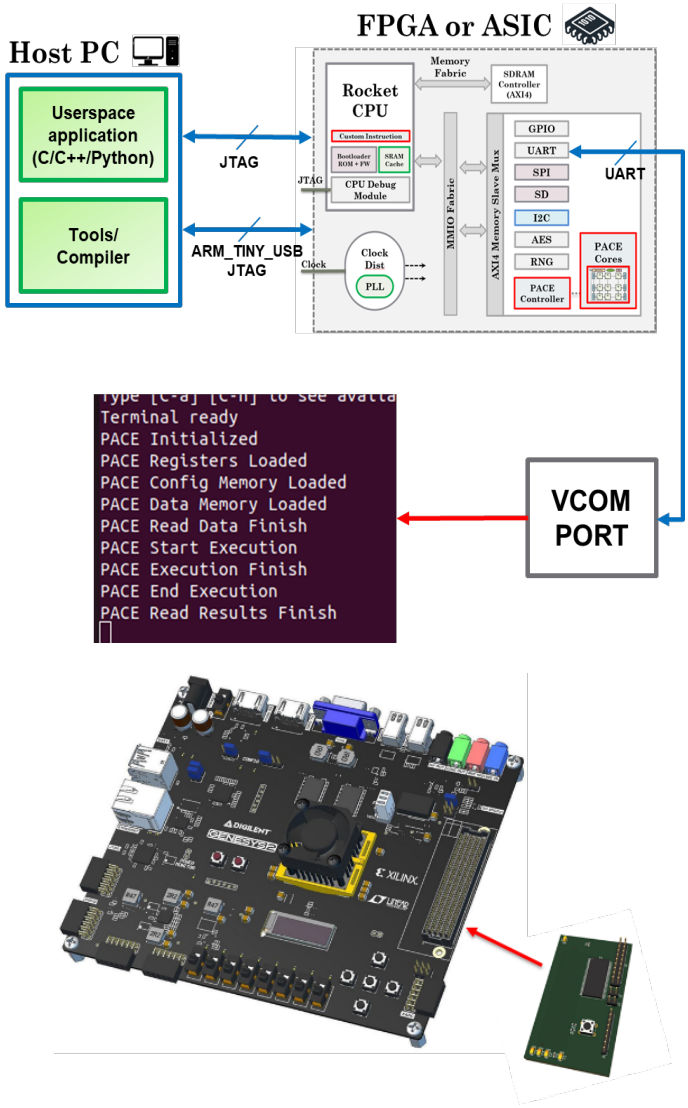


Figure 15 (a) FPGA Emulation of PACE Design (b) Digilent Genesys 2 FPGA

Progress towards PACE 2.0 Accelerator Chip

The PACE CGRA is designed in 12nm achieving the target ~2TOPS/W with a target frequency of 1GHz.

A Single-Input-Multiple-Data(SIMD) architecture is employed for higher throughput and efficiency. It outperforms state-of-the-art by 38%.

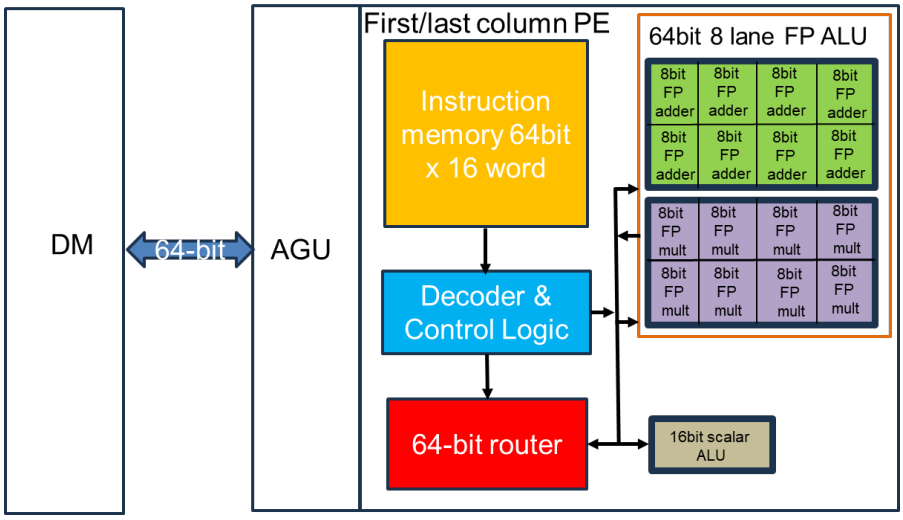
	Transpire [1]	[2]	**This work**
Year	2020	2020	**2025**
Tech(nm)	28	28	**12**
CGRA area(um2)	174230	122692	**361120**
# of PEs	64	64	**64**

Operand size	FP16/FP8	FP8/4	**FP8**
Voltage(V)	0.6	0.6	**0.8**
Frequency (MHz)	50	50	**1000**
Throughput (GFLOPS)	-	0.025	**512**
Power(mW)	-	0.162	**260**
Efficiency(GFLOPS/W)	224	156	**1966**
Memory (KB)	36	36	**80**
Norm. area	0.592	0.416	**0.364**
Norm efficiency	1219	849	**1966**
Stand out feature	SIMD, trans-precision computing		**SIMD Float8 and SISD Int16 ops with Address Generation Unit**

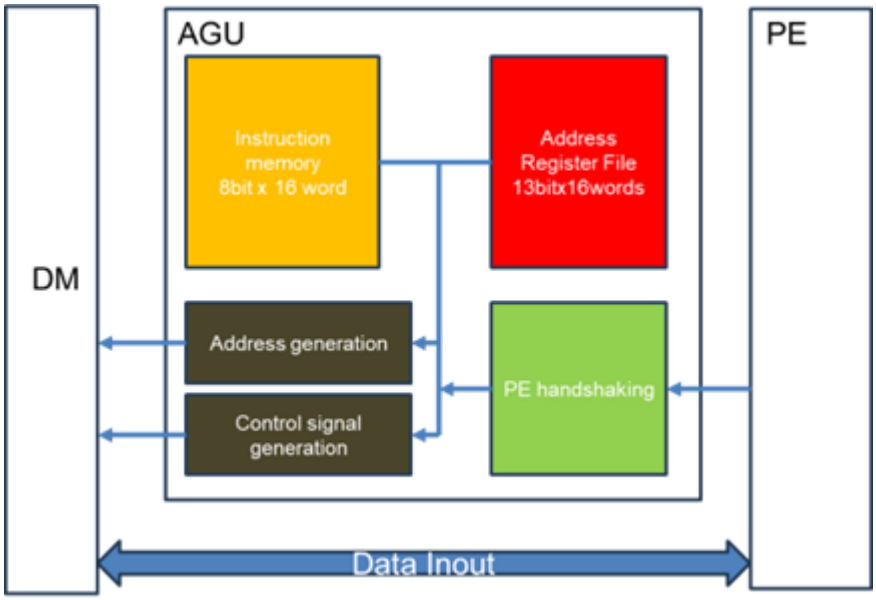
$$\text{Norm. efficiency} = \text{efficiency} \times \left(\frac{\text{node}}{40\text{nm}}\right)^2 \qquad \text{Norm. area} = \text{area} \times \frac{40\text{nm}}{\text{node}}$$

CGRA overview (Key features in the PACE CGRA)

To improve power efficiency and manage diverse AI workloads requiring higher throughput, the PE now supports 8-bit eight lane SIMD operations, thereby increasing the Data-memory (DM) and Processing-element (PE) data-width from earlier 16bit to 64bit. Each PE can perform eight parallel 8-bit floating point addition or multiplication, increasing peak throughput to 0.512TOPS and peak efficiency to 1.96TOPS/W at nominal 0.8V and 1GHz operating speed.



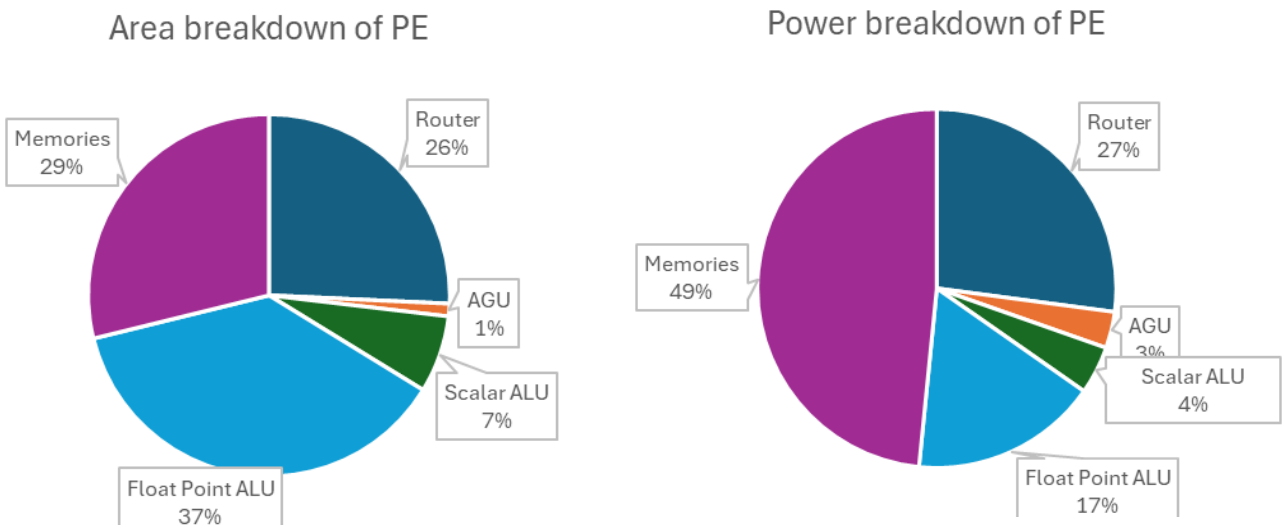
Block diagram of SIMD enabled processing element(PE) with multiple lane float operation support and integrated Address Generation Unit(AGU).



Block diagram of AGU logically coupled between the PE and Data Memory (DM)

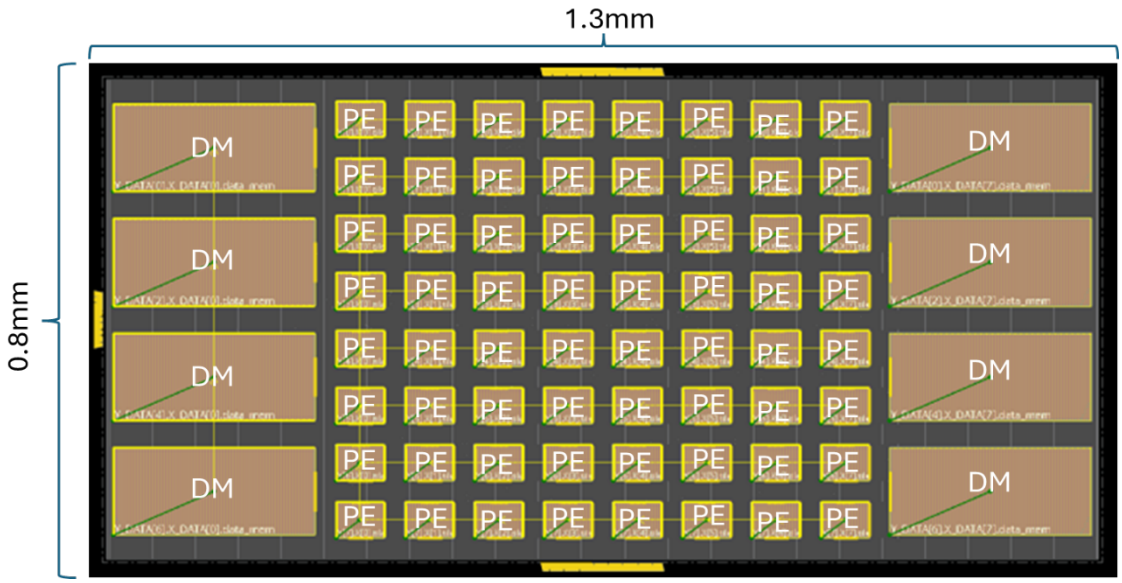
The first and last column PEs that access the memory, now have an Address-Generation-Unit (AGU), which helps to reduce the number of cycles needed to get the output. Approximately 75% of the PE instructions due to Load/store address calculation are now performed by the AGU in parallel. This translates to 75% improvement in initiation interval (II).

The AGU stores Load/Store instructions in its Config Memory (CM) and address in the Address Register File (ARF). This gives the AGU the capability to perform addressing of the DM in both stride and constant address modes.



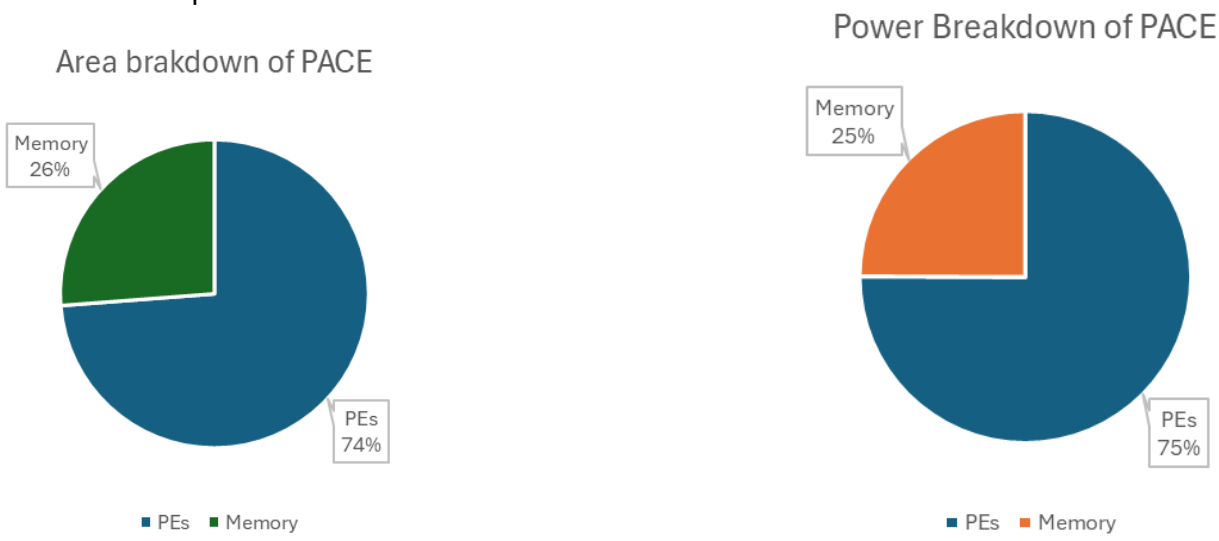
Area and power breakdown of the PE

First-cut layout of PACE 2.0 chip:



Layout of PACE chip with 8x8 PE array and eight banks of Data Memory (DM)

The PACE layout occupies a total of 1.04mm2 area. Below is a breakup for area and power for the PACE chip:



Area and power breakdown of the PACE

B) Progress towards PACE Compiler

As PACE is a software-define hardware accelerator, it requires scalable compiler to generate high-quality mapping of applications specified in high-level programming languages such as C/C++ onto the CGRA accelerator. We have made tremedous progress in developing state-of-the-art compiler for CGRA accelerators. *Our Morpher compiler in the best in class in the world in*

generating quality mapping, achieving fast compilation time, offering flexibility to explore different architectural designs, and providing end-to-end design flow including architecture specification, customized compiler generation, hardware synthesis, emulation, and verification/validation. We have made Morpher available as open-source project in December 2022 and various research groups across the world have reached out to us for collaborations on this toolchain.
https://github.com/ecolab-nus/morpher

Morpher [12] is a powerful, integrated compilation and simulation framework, that can assist design space exploration and application-level developments of CGRA based systems. Morpher can take an application with a compute intensive kernel as input, compile the kernel onto a user-provided CGRA architecture, and automatically validate the compiled kernels through cycle-accurate simulation using test data extracted from the application. Morpher can handle real-world application kernels without being limited to simple toy kernels through its feature-rich compiler. Morpher architecture description language lets users easily specify architectural features such as complex interconnects, multi-hop routing, and memory organizations.

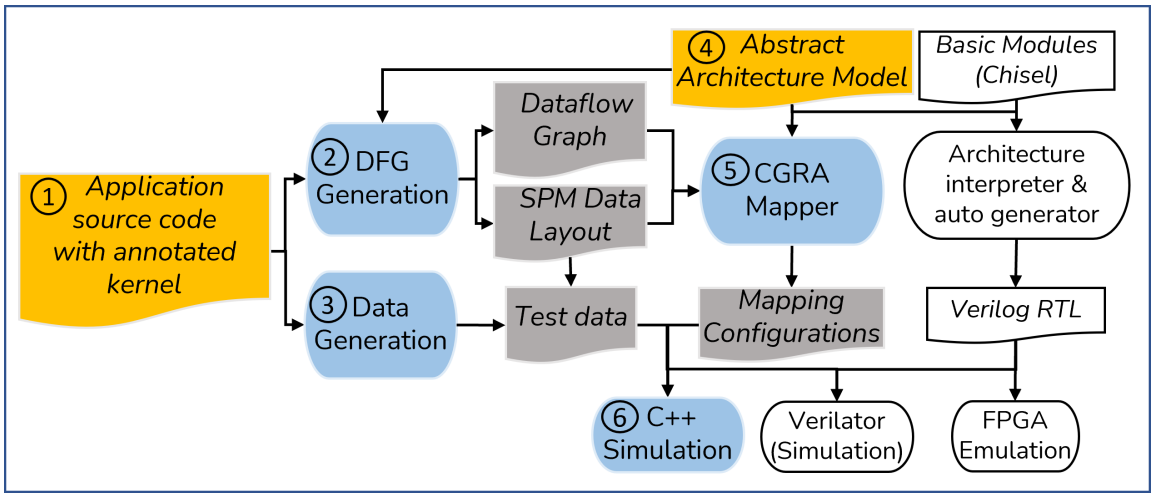


Figure 16. Morpher: An Open-Source Tool for CGRA Accelerators

Figure 17 compares Morpher with the existing state-of-the-art CGRA compilation frameworks.

Features		CGRA-ME	Pillars	Open CGRA	CCF	DSAGEN	ABC
DFG Generation	Models control divergence	✗	✗	✓	✓	✗	✓
	Recurrence edges	✗	✗	✓	✓	✗	✓
Architecture Modeling	Adapt user defined architectures	✓	✓	✓	✗	✓	✓
	Complex interconnects	✗	✗	✗	✗	✗	✓
	Different memory organizations	✗	✗	✓	✗	✗	✓
P&R Mapper	Architecture adaptive mapping	✓	✓	✗	✗	✓	✓
	Data layout aware mapping	✗	✗	✗	✗	✓	✓
	Recurrence aware mapping	✗	✗	✓	✓	✓	✓
Hardware Generation	Generate RTL	✓	✓	✓	✗	✓	✓
	Infer control path	✗	✓	✓	✗	✓	✓
	Infer multiplexers	✗	✗	✗	✗	✗	✓
Simulation & Validation	Cycle accurate simulation	✗	✓	✓	✓	✓	✓
	Test data generation	✗	✗	✗	✗	✗	✓
	Validation against test data	✗	✗	✗	✗	✗	✓

Figure 17. Comparison of Morpher (named ABC in this table) with State-of-the-Art toolchains

Morpher is enabled by underlying compilation technology that generates quality mapping but at 13x faster compilation time compared to existing approaches. We now describe the technology behind Morpher.

B.1) Hierarchical Mapping Approach

We have developed two hierarchical mapping approaches to improve the compilation time. The first is PANORAMA [1] a divide and conquer approach for mapping complex loop kernels on CGRA. Mapping complex loop kernels onto the CGRA becomes challenging mainly due to the irregular data dependencies in complex loop kernels and limited routing resources on the CGRA fabric. PANORAMA is a fast, scalable and portable compiler based on a divide-and-conquer approach to generate quality mapping for complex Dataflow Graphs (DFGs) onto larger CGRA. The higher-level mapping partitions the dataflow graph onto the CGRA clusters and guides the lower- level mapping, reducing overall complexity (Figure 18). PANORAMA improves the throughput of mapped complex loops kernels with large and irregular DFGs by up to 2.6x with 8.7x faster compilation time compared to the state-of-the-art techniques (Figure 19-20).

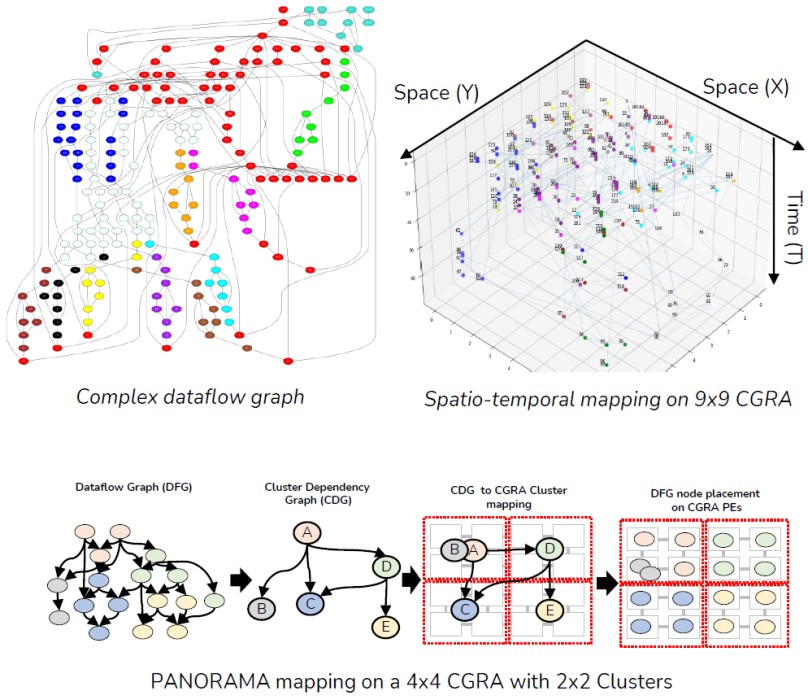


Figure 18: PANORAM hierarchical mapping for complex loop kernels on CGRA

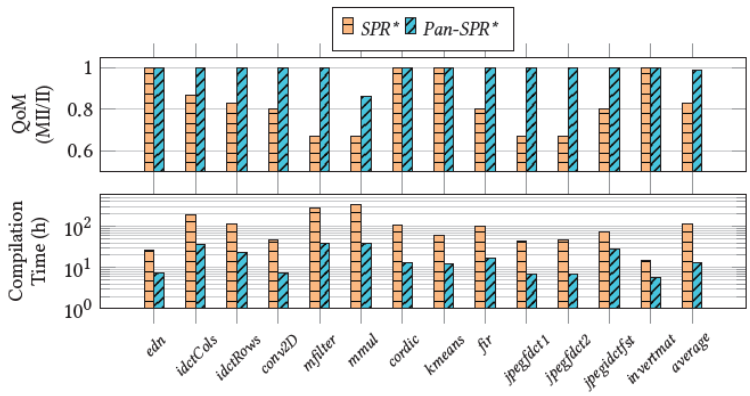


Figure 19: Comparison of quality of mapping and compilation time of PANORAMA with state-of-the-art CGRA compiler technique SPR

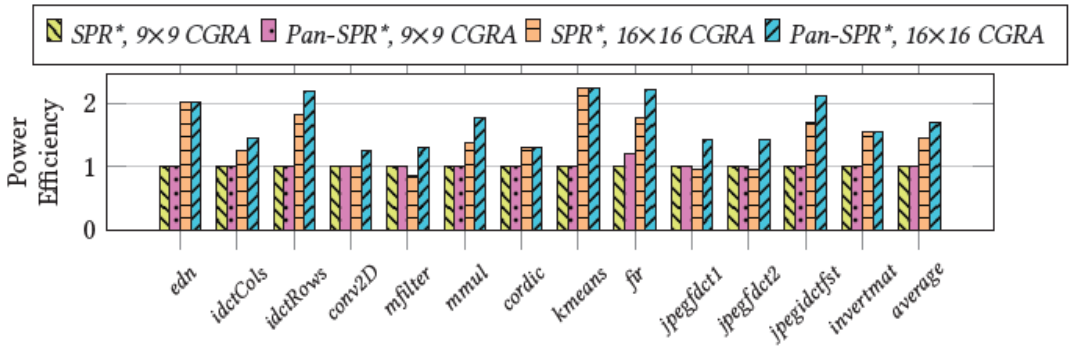


Figure 20: Comparison of power efficiency of the generated mappings of PANORAMA with state-of-the-art CGRA compiler technique SPR

While Panorama handles complex loop kernels, HiMap [9] is adept at producing close to optimal solutions for regular computational kernels prevalent in existing and emerging application domains on larger CGRAs. The state-of-the-art compiler technology falls short in generating high-performance mapping within an acceptable compilation time, especially with increasing CGRA size (beyond 4x4 PE array).

The key strategy behind HiMap's efficiency and scalability is to exploit the regularity in the computation by employing a virtual systolic array as an intermediate abstraction layer in a hierarchical mapping. HiMap first maps the loop iterations of the kernel onto a virtual systolic array and then distills out the unique patterns in the mapping. These unique patterns are subsequently mapped onto sub-spaces of the physical CGRA. They are arranged together according to the systolic array mapping to create a complete mapping of the kernel (Figure 21).

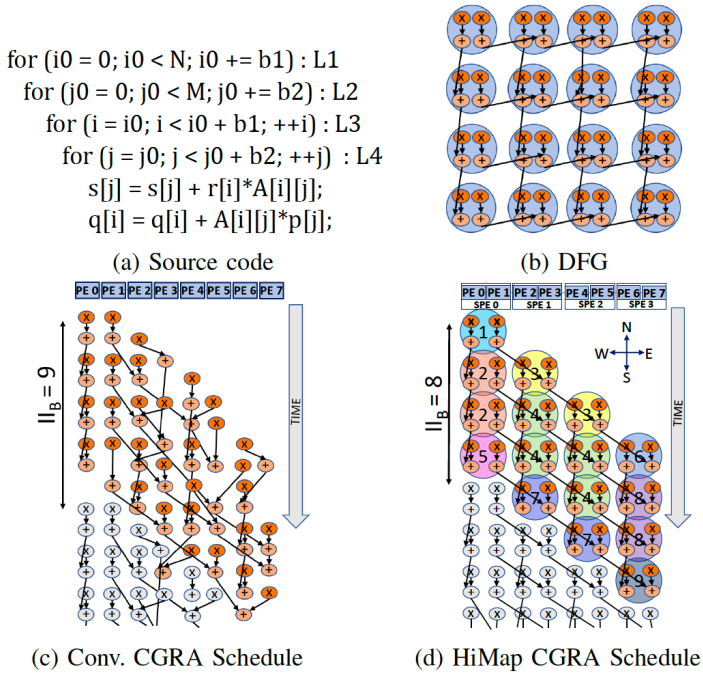


Figure 21: Illustration of HiMap versus conventional CGRA mapping approach.

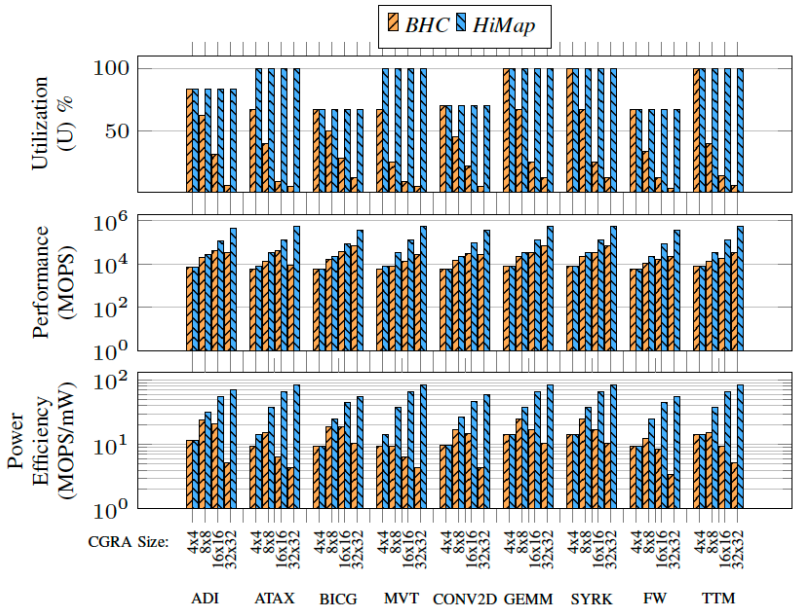


Figure 22: Comparison of quality of mapping generated by HiMap against state-of-the-art CGRA compiler technique BHC (best of CGRA-ME and traditional simulated annealing)

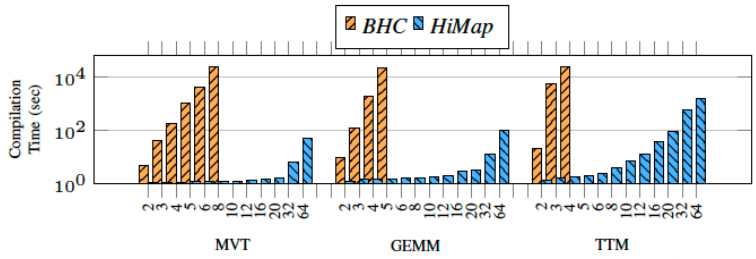


Figure 23: Comparison of compilation time of HiMap against state-of-the-art CGRA compiler technique BHC (best of CGRA-ME and traditional simulated annealing)

Experimental results show that HiMap can generate application mappings that hit the performance envelope of the CGRA. HiMap offers 17.3x and 5x improvement in performance and energy efficiency of the mappings compared to the state-of-the-art (Figure 22). The compilation time of HiMap for near-optimal mappings is less than 15 minutes for 64x64 CGRA, while existing approaches take days to generate inferior mappings (Figure 23).

B.1) Portable Compilation Framework

While PANOMA and HiMap employ hierarchical mapping to improve the mapping quality and significantly reduce the compilation time, we still need a good-quality low-level mapper/compiler. Currently, the mappers/compilers like SPR are handcrafted for CGRAs, which is challenging specially when we develop heterogeneous CGRA accelerator with techniques like REVAMP [3]. Therefore, we designed a portable compilation framework, called LISA [2], that can be tuned automatically to generate quality mapping for varied spatial accelerators. Our key contribution is to automatically identify the impact of the dataflow graph (DFG) structure characteristics (representing an application) on the mapping for a new accelerator. Towards this end, we abstract the DFG structure in graph attributes, use Graph Neural Network (GNN) to analyze the graph attributes, and identify the mapping impact for an accelerator architecture with an all-encompassing global view. Finally, we augment a simulated annealing-based mapping approach to account for the impact of DFG structure in guiding the placement of the dataflow graph nodes and the routing of the dependencies on the accelerator (Figure 24). Our experimental

evaluation concretely demonstrates the substantial benefit of our approach compared to the state-of-the-art solutions. LISA achieves 594x and 17x compilation time reduction compared to ILP (Integer Linear Programming) and SA (Simulated Annealing) used in SPR. For 71 combinations of benchmarks and accelerators, LISA can map all of them while ILP cannot map 48 combinations and SA cannot map 21 combinations.

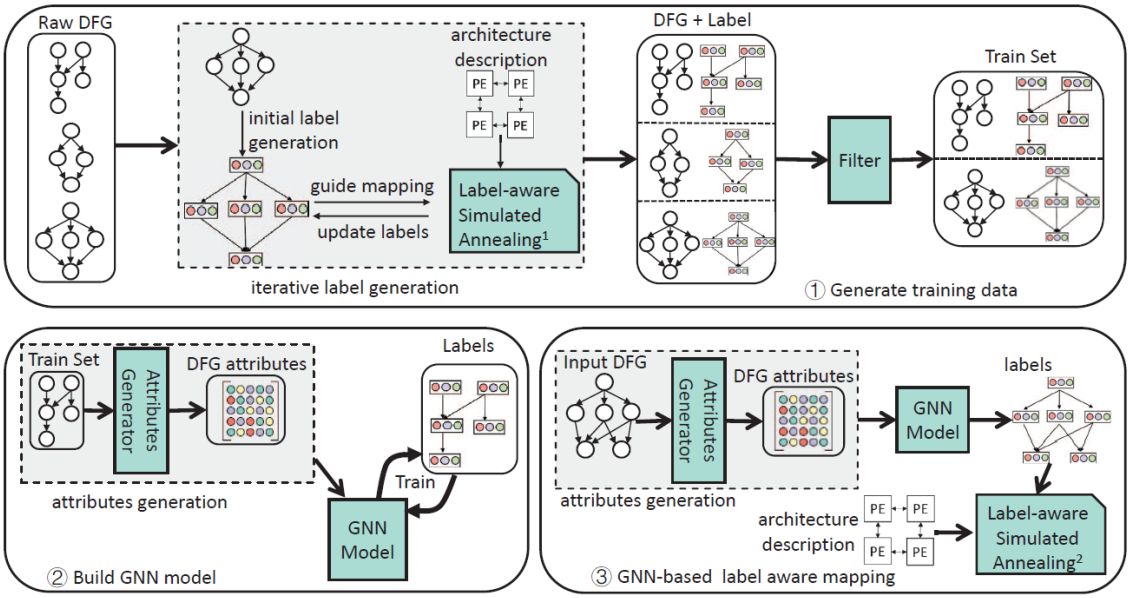


Figure 24: LISA Framework for Portable and Quality Compilation on CGRAs

C) PACE 2.0 Architectural Innovations

C.1) Heterogeneous CGRA Architecture

The key distinguishing factor of the final PACE v2.0 architecture is that it will be heterogenous CGRA rather than homogeneous.

Our current PACE 1.0 chip is homogeneous. Indeed, most CGRAs adhere to a canonical structure where a homogeneous set of processing elements and memories communicate through a regular interconnect due to the simplicity of the design. Unfortunately, the homogeneity leads to substantial idle resources while mapping irregular applications and creates inefficiency as shown in the Table below for three homogeneous CGRAs (ADRES, HyCUBE, and Softbrain).

	HM-ADRES	HM-HyCUBE	HM-Softbrain
Compute Utilization	16.5%	28.2%	26.8%
Interconnect Utilization	1.6%	8.8%	13.1%
Valid Configurations	12.5%	21.5%	-

We have mitigated the inefficiency by systematically and judiciously introducing heterogeneity in CGRAs in tandem with appropriate compiler support. We have proposed REVAMP [3], an automated design space exploration framework that helps architects uncover and add pertinent heterogeneity to a diverse range of originally homogeneous CGRAs when fed with a suite of target applications. The REVAMP framework is shown in Figure 25. REVAMP explores a comprehensive set of optimizations encompassing compute, network, and memory heterogeneity, thereby converting a uniform CGRA into a more irregular architecture with improved energy efficiency.

As CGRAs are inherently software scheduled, any micro-architectural optimizations need to be partnered with corresponding compiler support, which is challenging with heterogeneity. The

REVAMP framework extends compiler support for efficient mapping of loop kernels on the derived heterogeneous CGRA architectures.

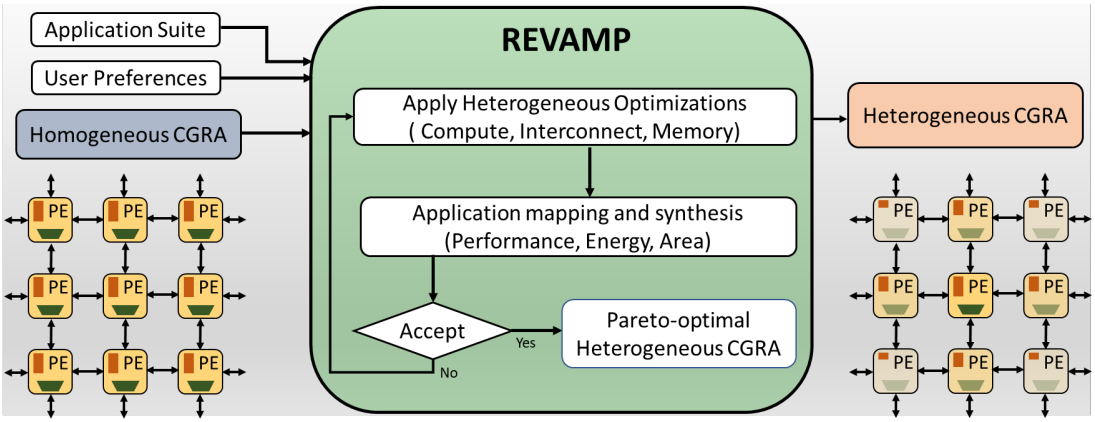


Figure 25. REVAMP Framework

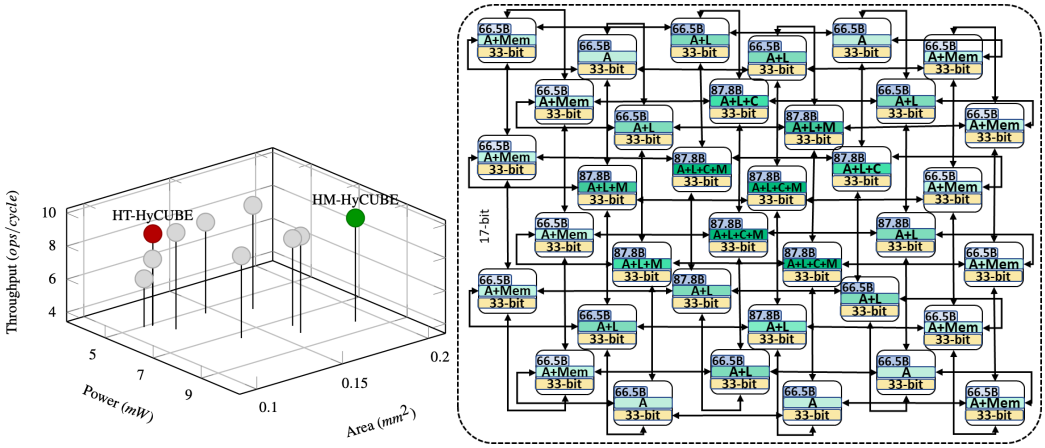


Figure 26. Deriving heterogeneous HyCUBE (HT-HyCUBE on the right) from the homogeneous HyCUBE (HM_HyCUBE)

We showcase REVAMP on three state-of-the-art homogeneous CGRAs, demonstrating how REVAMP derives a heterogeneous variant of each homogeneous architecture, with its corresponding compiler optimizations. Figure 26 shows the automated derivation of a heterogeneous HyCUBE CGRA architecture from a homogeneous HyCUBE. Our results (Figure 27-29) show that the derived heterogeneous architectures achieve up to 52.4% power reduction, 38.1% area reduction, and 36% average energy reduction over the corresponding homogeneous versions with minimal performance impact for the selected kernel suite.

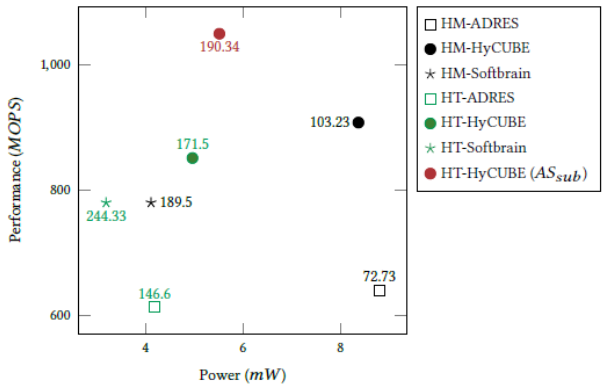


Figure 27: Power-Performance comparison of homogeneous and heterogeneous CGRA

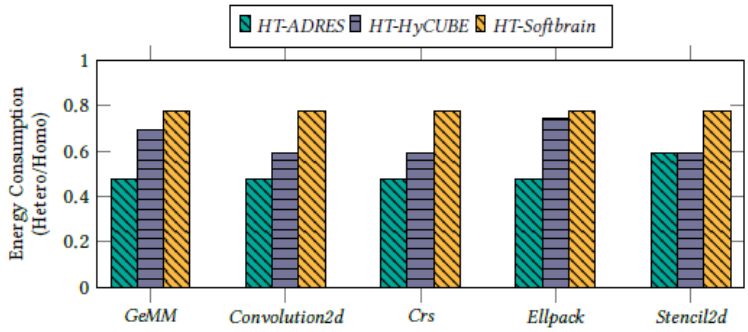


Figure 28: Energy comparison of homogeneous and heterogeneous CGRA

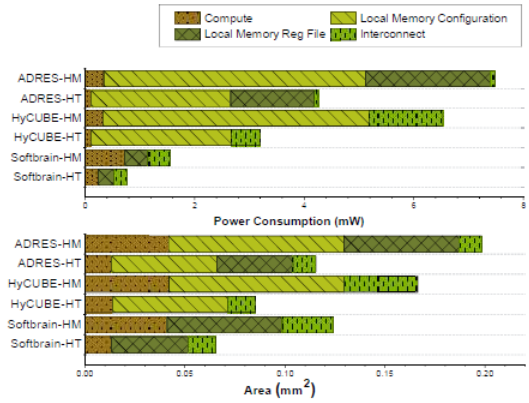


Figure 29: Energy comparison of homogeneous and heterogeneous CGRA

C.2) Flexible CGRA with Spatio-Temporal Vector Dataflow

CGRAs like HyCUBE and PACE 1.0 tend to follow a fixed execution model, either spatio-temporal execution or spatial execution, leading to an imbalance in internal resource distribution and limiting efficiency. Spatio-temporal execution with the highest reconfiguration frequency wastes the most energy on re-configuring the CGRA. In contrast, spatial execution with the least reconfiguration degrades the performance and spends the most energy on unwanted data movements. We propose FLEX [19] which bestows a flexible CGRA execution paradigm with novel spatiotemporal vector dataflow execution. In spatio-temporal vector dataflow, a vector of data is processed in sequence, reducing the reconfiguration frequency significantly. The proposed execution model balances the internal resource distribution by significantly reducing the reconfiguration energy compared to the spatio-temporal execution and averting performance and data movement impacts of spatial execution.

The flexibility introduced with FLEX covers a complete spectrum of reconfiguration frequencies starting from spatio-temporal execution to spatial execution through multiple vector sizes of spatio-temporal vector dataflow. FLEX can be used as a design space exploration framework to choose the best execution mode/vector size for individual application kernels based on the requirements. Figure 30 shows an overview of FLEX.

We evaluate FLEX against a generic spatio-temporal CGRA (HyCUBE) and a spatial CGRA (SNAFU from CMU), showing that spatio-temporal vector dataflow execution achieves the highest efficiency in most application kernels. FLEX achieves ~ 53% and ~ 38% improvement in average power efficiency compared to spatio-temporal (PACE 1.0) and spatial (SNAFU from CMU) baselines, respectively, with nearly the same area budget.

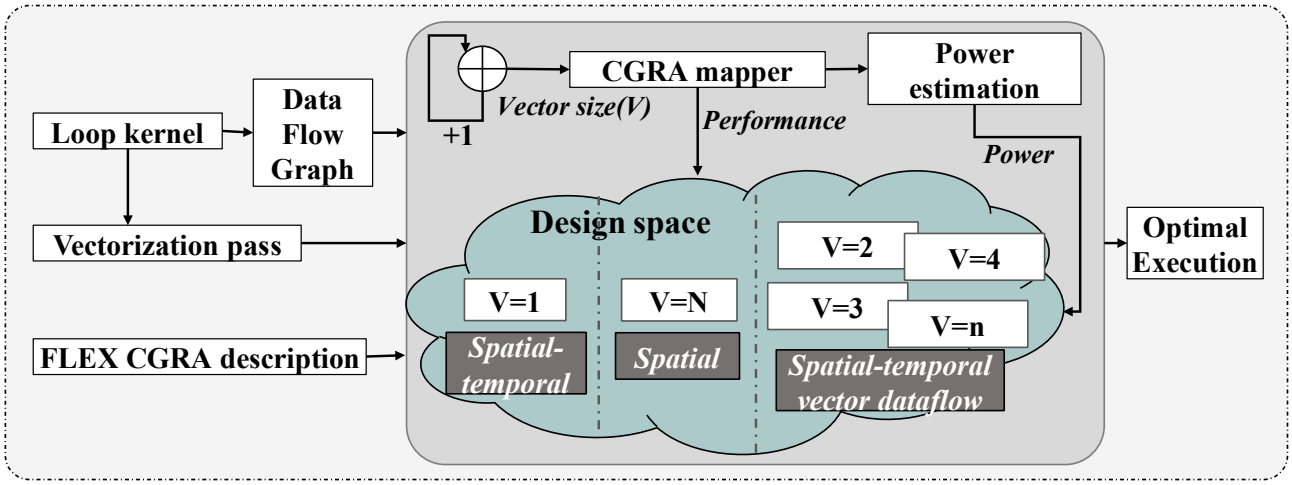


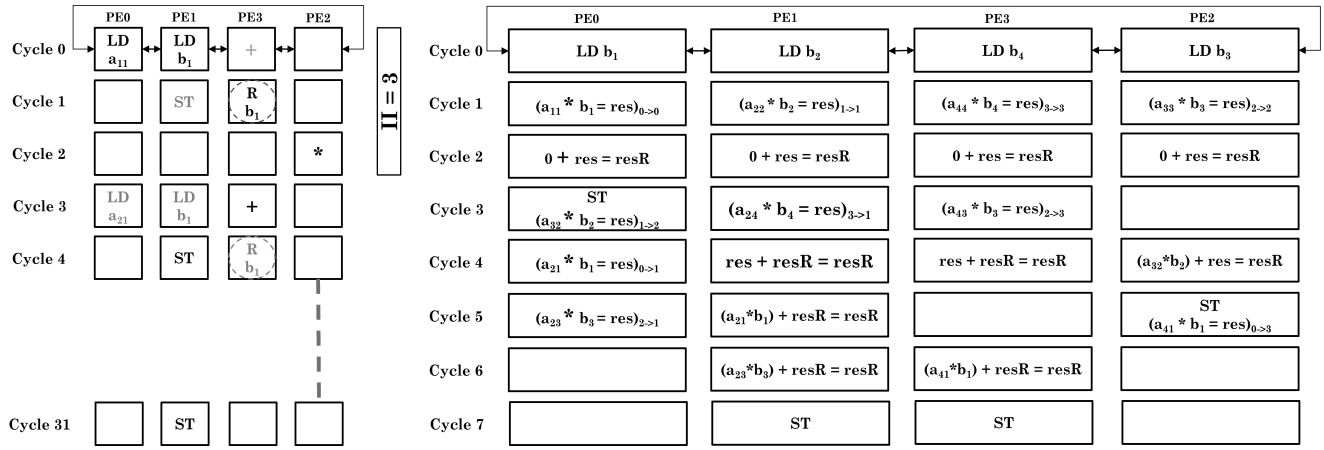
Figure 30: FLEX Architecture covering the entire spectrum from spatial to spatio-temporal

C.3) Active Message CGRA for Irregular Applications

The specialization for regular control flow and memory accesses in CGRAs limit their effectiveness in executing irregular workloads, such as sparse linear algebra and graph analytics with irregular control flow and memory accesses. To address this limitation, we propose Active Message CGRA (or AM-CGRA [22]), an architecture and compilation technique to execute irregular applications efficiently.

AM-CGRA sends messages across the fabric that dynamically deploys and executes instructions on idle processing elements (PEs). Thus, unlike traditional architecture with static instructions within each PE, AM-CGRA brings control to idle PEs. The architecture distributes the application's data and control across multiple PEs, and load balances by executing messages enroute. Our initial experiments demonstrate that AM-CGRA significantly improves performance, achieving 4x and 1.75x speedup over traditional shared-memory and distributed-memory CGRA architectures, respectively, with the same power budget and area. It also achieves PE array utilization of 78.6%, compared to 45% and 55.4% for traditional shared memory and distributed memory CGRAs, respectively.

Figure 31 illustrates the cycle-by-cycle progression of the AM-CGRA architecture versus standard CGRA architecture for sparse-matrix vector multiplication (SpMV). In the figure, each $(instr)_{src->dest}$ represents an AM (active message) that contains the instruction and operands being executed at an idle PE, while the numeric values outside denote the source and destination of the messages. The output of this execution is then utilized in the ongoing execution at the destination PE.



(a) Static CGRA Execution Model (b) AM-CGRA Execution Model

Figure 31: Traditional CGRA vs. Active Message CGRA

C.4) Data-Centric CGRA Accelerator

Classic CGRAs statically map compute operations onto the processing elements (PE) and route the data dependencies among the operations through the Network-on-Chip. However, CGRAs are designed for fine-grained static instruction-level parallelism and struggle to accelerate applications with dynamic and irregular data-level parallelism, such as graph processing. To address this limitation, we present Flip [20], a novel accelerator that enhances traditional CGRA architectures to boost the performance of graph applications. Flip retains the classic CGRA execution scheme while introducing a divergent mode for efficient graph processing. Specifically, it exploits the natural data parallelism of graph algorithms by mapping graph vertices onto processing elements (PEs) rather than the operations and supporting dynamic routing of temporary data according to the runtime evolution of the graph frontier (Figure 32-33). As Flip targets embedded systems with stringent energy and area budgets, limited memory inside each PE poses a unique challenge of reconciling two conflicting goals: reducing communication by capturing data locality and avoiding congestion by breaking clusters. Therefore, an original compiler is designed. Experimental results demonstrate that Flip achieves up to 36x speedup with merely 19% more area compared to classic CGRAs. Compared to state-of-the-art large-scale graph processors, Flip has similar energy efficiency and 2.2x better area efficiency at a much-reduced power/area budget.

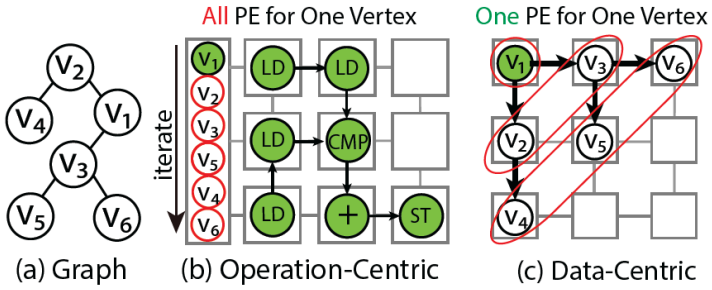


Figure 32: Operation-centric vs. Data-centric execution modes. Operation-centric mode maps the operations on the PEs and processes the vertices stored in scratchpad memory iteratively. Our proposed data-centric mode maps the vertices on different PEs and processes them in parallel with identical code.

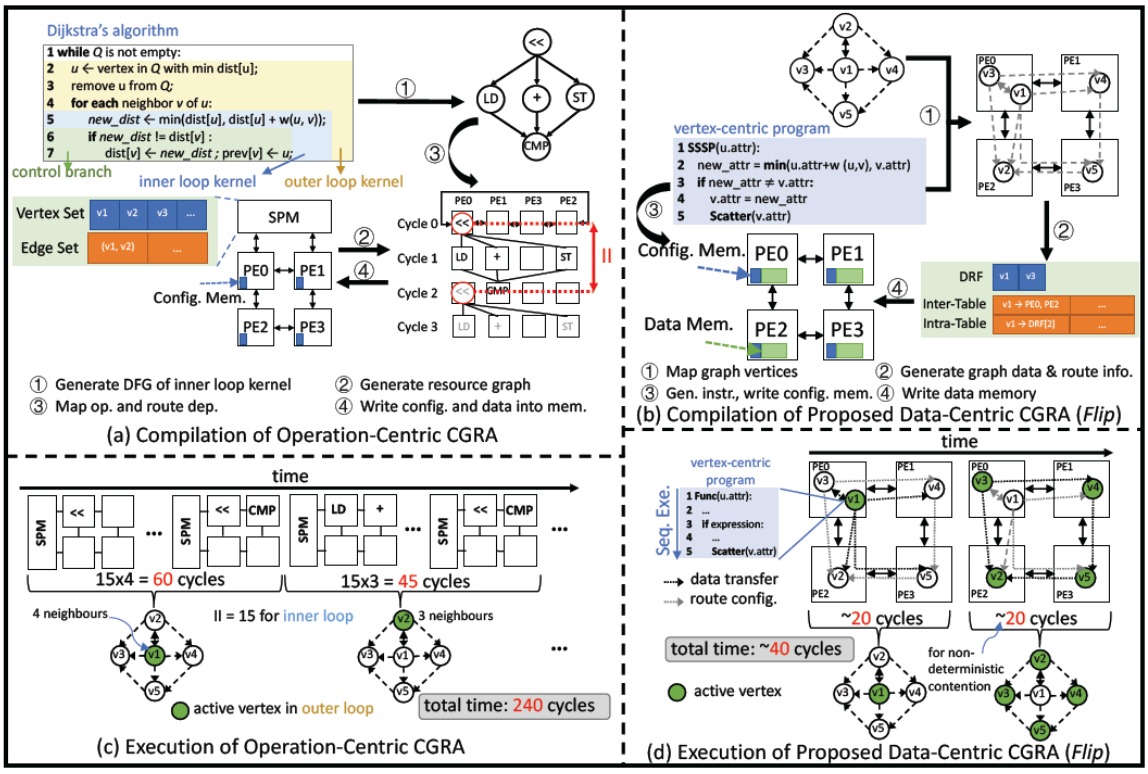


Figure 33: Compilation and execution for operation-centric CGRA and data-centric CGRA, taking Single Source Shortest Path (SSSP) problem as an example. Data-centric CGRA benefits from data-level parallelism and outperforms operation-centric CGRA which suffers from low instruction-level parallelism in graph algorithms.

C.5) Transformer Acceleration using in-situ computing

In our work "ASADI: Accelerating Sparse Attention using Diagonal-based In-situ Computing," [21] we address the performance bottleneck of the self-attention mechanism in Transformer-based language models, particularly for long sequences. Transformers have revolutionized Natural Language Processing (NLP) and Computer Vision (CV) due to their ability to model long-range dependencies through self-attention mechanisms. However, the quadratic complexity of self-attention with respect to the number of tokens results in high computational overhead, especially for long sequences.

We observe that sparse attention, while effective in reducing computational complexity by eliminating weak connections between tokens, introduces significant random access overhead. This overhead limits the computational efficiency of existing sparse attention mechanisms. To overcome this challenge, we propose ASADI, a novel software-hardware co-designed sparse attention accelerator. On the software side, we develop a new sparse matrix computation paradigm that directly supports the diagonal (DIA) format, which we identified as having excellent diagonal locality. On the hardware side, we introduce an innovative sparse attention accelerator that efficiently implements this DIA-based computation paradigm using highly parallel in-situ computing with ReRAM.

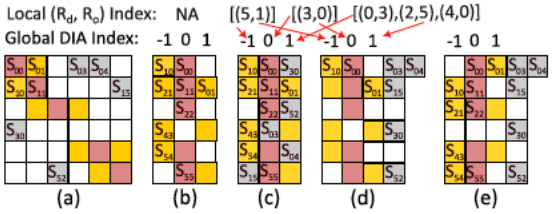


Fig. 7. (a) Sparse S matrix with bubbles, (b) Bubble-free DIA compression, (c) Bubble-containing DIA compression, (d) Decompress non-central diagonals, (e) Decompress central diagonals

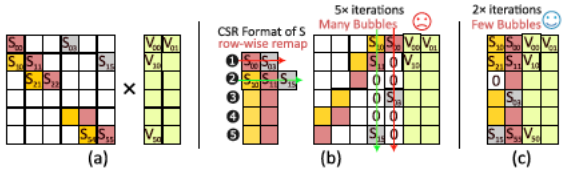
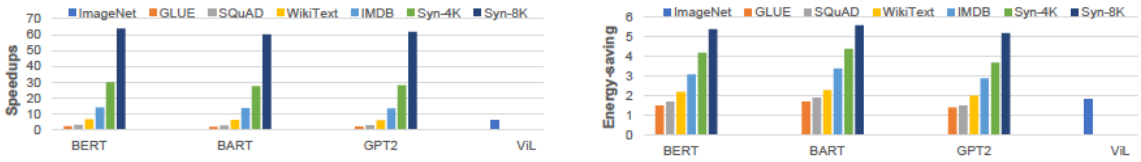


Fig. 9. (a) Matrix multiplication between sparse S matrix and dense V matrix, (b) In-situ computing with CSR format of matrix S, (c) In-situ computing with DIA format of matrix S

Diagonal matrix format (DIA) for sparse attention acceleration

Our experimental results demonstrate that ASADI achieves an average performance improvement of 18.6× and energy savings of 2.9× compared to a processing-in-memory (PIM) based baseline. By leveraging diagonal locality and utilizing a comprehensive hardware-software co-design, ASADI significantly enhances the performance and energy efficiency of Transformer-based models, especially for applications involving long sequences.

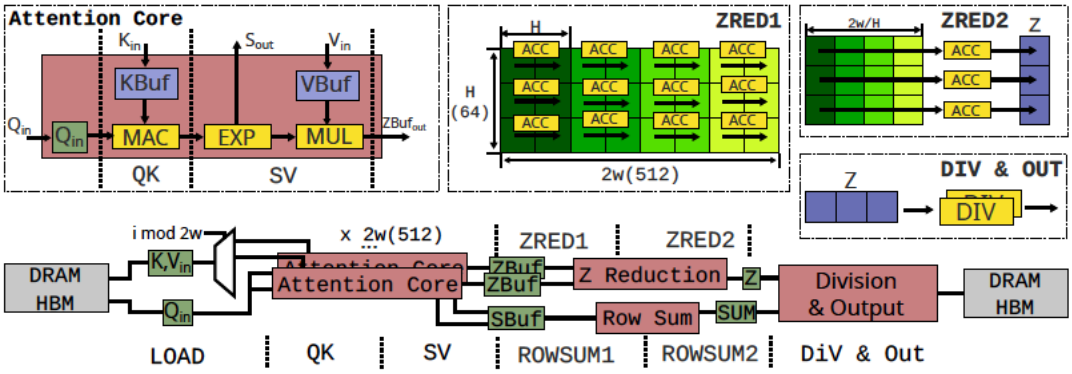


Performance and energy efficiency comparison between ASADI and PIM Baseline

C.6) Transformer Acceleration on reconfigurable computing

In our paper "SWAT: Scalable and Efficient Window Attention-based Transformers Acceleration on FPGAs," [24] we address the challenge of efficiently supporting long context lengths in Transformer models on reconfigurable computing devices such as FPGAs. Traditional Transformer models suffer from quadratic complexity in their self-attention computation as mentioned earlier, making them inefficient for long sequences. Sliding window-based static sparse attention mitigates this issue by limiting the attention scope of input tokens, reducing theoretical complexity from quadratic to linear. However, this approach does not align perfectly with the microarchitecture of conventional accelerators, leading to suboptimal implementation.

To tackle this, we propose SWAT, a dataflow-aware FPGA-based accelerator that efficiently leverages the structured sparsity of window attention. Our design maximizes data reuse by combining row-wise dataflow, kernel fusion optimization, and an input-stationary design that takes advantage of the distributed memory and computation resources of FPGAs. This allows us to achieve significant improvements in latency and energy efficiency. Experimental results show that SWAT achieves up to 22× and 5.7× improvements in latency and energy efficiency, respectively, compared to a baseline FPGA-based accelerator, and up to 15× energy efficiency improvement compared to GPU-based solutions. By deeply analyzing the dataflow and optimizing the microarchitecture for window attention, SWAT provides a scalable and efficient solution for accelerating Transformer models on FPGAs.



SWAT Microarchitecture

Execution time, memory usage, and energy efficiency comparison of SWAT with state-of-the-art

In our paper “ZeD: A Generalized Accelerator for Variably Sparse Matrix Computations in ML” [27], we focus on handling sparse ML workload through hardware-software codesigned approach.

Modern Machine Learning (ML) models employ sparsity to mitigate storage and computation costs; but it gives rise to irregular and unstructured sparse matrix operations that dominate the execution time and require specialized accelerators to meet the performance and energy targets. Contemporary sparse matrix accelerators, optimized for extreme sparsity, frequently fall short in addressing the variable and moderate degrees of sparsity prevalent in most ML models. Variable sparsity leads to inefficiency in the storage and processing of matrices. In response to this challenge, we propose an adaptive and generalized architecture design, ZeD, capable of accommodating the variably sparse matrix computations in ML models. Our innovative design integrates a bit-tree compression format and zero-detection hardware, resulting in highly efficient packing, storage, retrieval, and processing of sparse matrices. Furthermore, we propose a matrix row reorganization strategy based on sparsity similarity to substantially enhance memory reuse. Synthesis results of ZeD demonstrate a 3.2×

improvement in performance per area over state-of-the-art solutions across a spectrum of ML workloads characterized by wide-ranging sparsities.

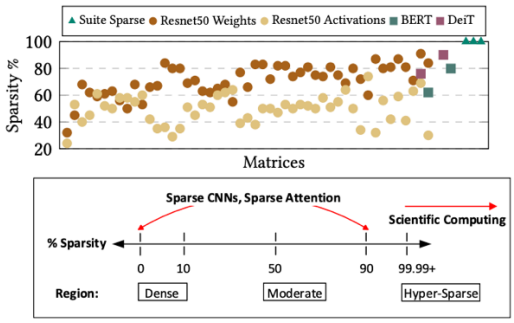


Figure 1: Variation in sparsity

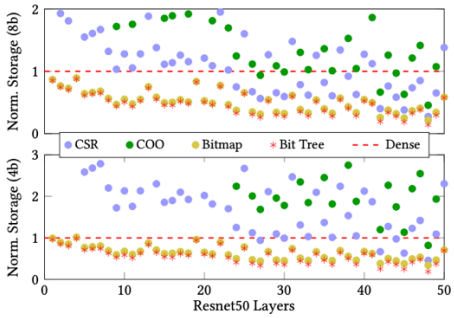


Figure 2: Comparison of storage costs of various sparse formats, normalized to the dense format (uncompressed) for 8-bit and 4-bit precisions.

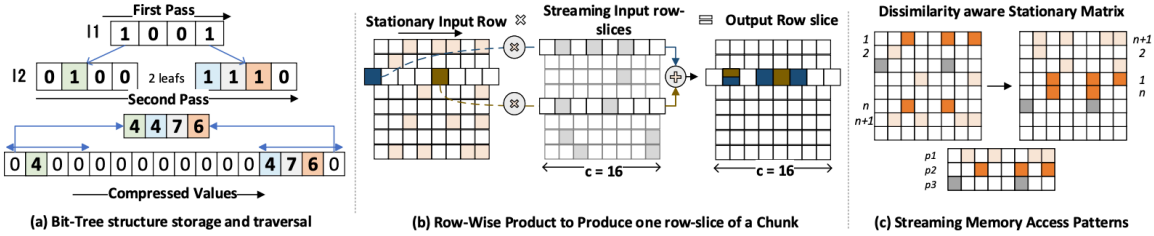


Figure 4: (a) Illustration of a 2-level bit-tree: 4 non-zeroes in a 16-element slice. (b) Working of row-wise product (c) Extracting Streaming Patterns from a Stationary Matrix

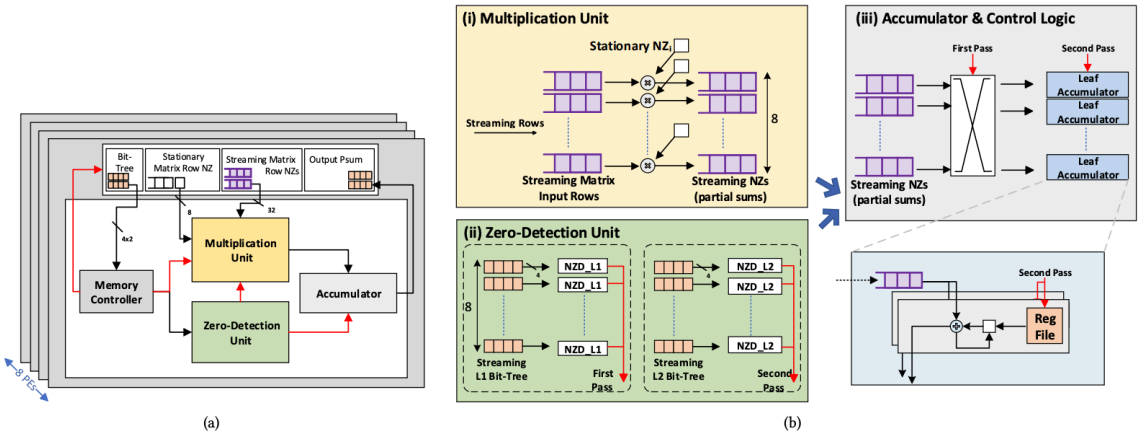


Figure 7: (a) Architectural Overview: datapath (black) & control flow (red) (b) Working of various components of a PE

C.7) DVFS for CGRA

In our paper "ICED: An Integrated CGRA Framework Enabling DVFS-Aware Acceleration", [26] we collaborate with Google and others to explore DVFS.

Coarse-grained reconfigurable arrays (CGRAs) are a promising solution to enable energy-efficient acceleration of applications from different domains. By leveraging reconfiguration at the functional level, they can adapt to significantly different computational patterns. However, the relationships of voltage and frequency with the utilization of CGRA resources

and the dynamic management of them are not well explored, leading to inefficient designs. CGRAs have also been successful in accelerating datadependent streaming applications. However, in these applications, the execution time of each kernel in the pipeline might dynamically vary depending on the characteristics of the input. This also leads to under-utilization of resources for the dynamically changing kernels that do not limit the application throughput. DVFS can also improve energy efficiency for these applications by dynamically changing the voltage and frequency levels of tiles that host non-performance-constraining kernels. This paper proposes ICED – an integrated DVFS-aware framework to map applications on CGRAs that support power islands. ICED proposes a CGRA architecture supporting DVFS islands at varying granularity (from a single tile to a group of tiles) and the related DVFS-aware compilation and mapping toolchain. ICED is the first work that introduces DVFS support for spatio-temporal CGRAs at power-island levels. The experimental evaluation shows that ICED improves average utilization by 2.3× and energy-efficiency by 1.32× over a conventional CGRA. With streaming applications, ICED can achieve up to 1.26× energyefficiency compared with a state-of-the-art CGRA that introduces partial dynamic reconfiguration to adapt to variations in kernels' throughput.

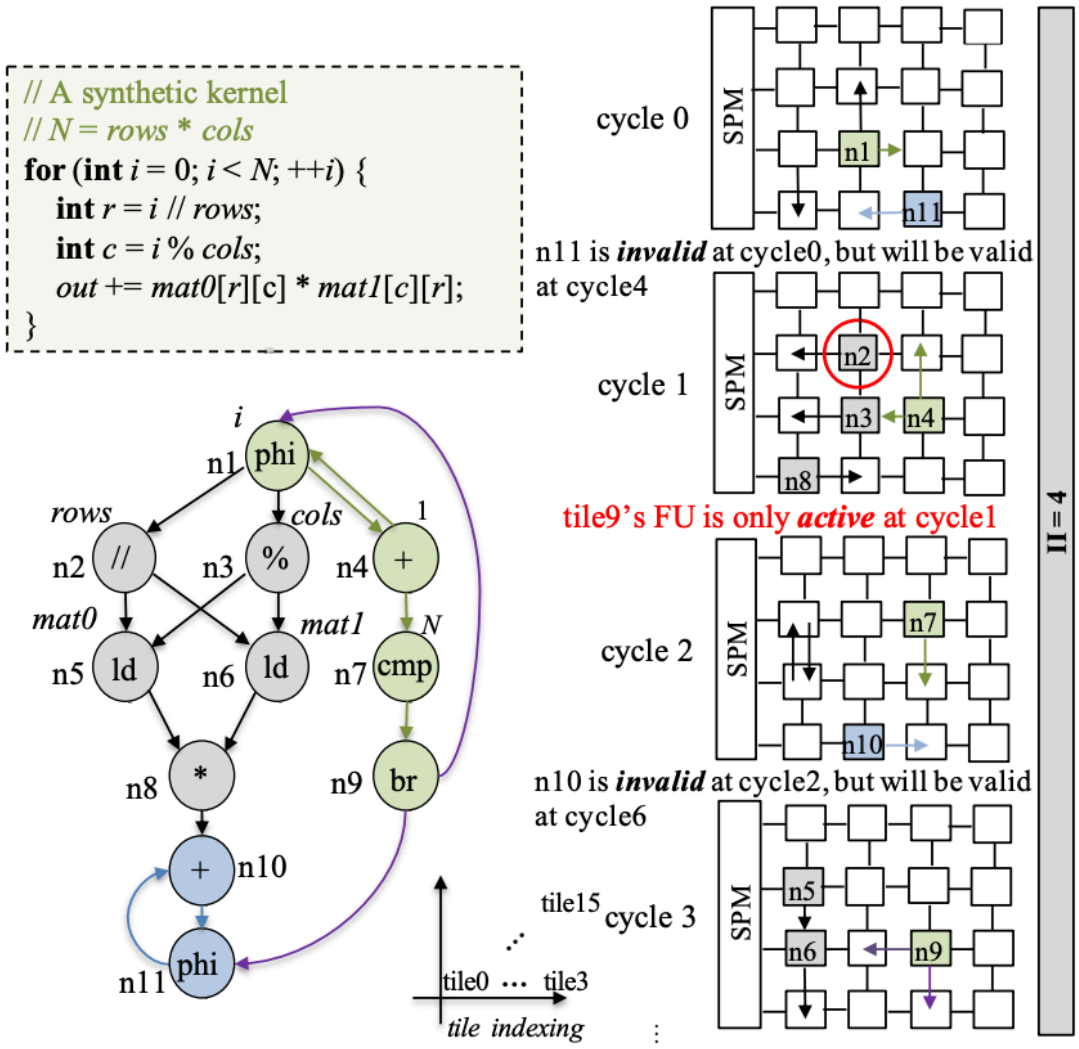


Fig. 1: A synthetic kernel, DFG, and the conventional mapping on a 4×4 CGRA – The DFG nodes `n1`, `n4`, `n7`, and `n9` determine the II, which are on the critical path and colored in green. The DFG nodes `n10` and `n11` forming another recurrence cycle are colored in blue. The left DFG nodes that are not on any critical path are colored in grey. The scheduling will repeat every II (i.e., 4) cycles. n5 has to be mapped on a leftmost tile as it is a load operation and only the leftmost tiles are connected to the scratchpad memory. In addition, `tile0` has `n8` on it at `cycle1`. However, `tile0` is also used for receiving/routing the data at `cycle0` and `cycle3`. Note that we use the 4×4 CGRA as a motivating example for simplification, the proposed ICED prototype is based on a 6×6 CGRA.

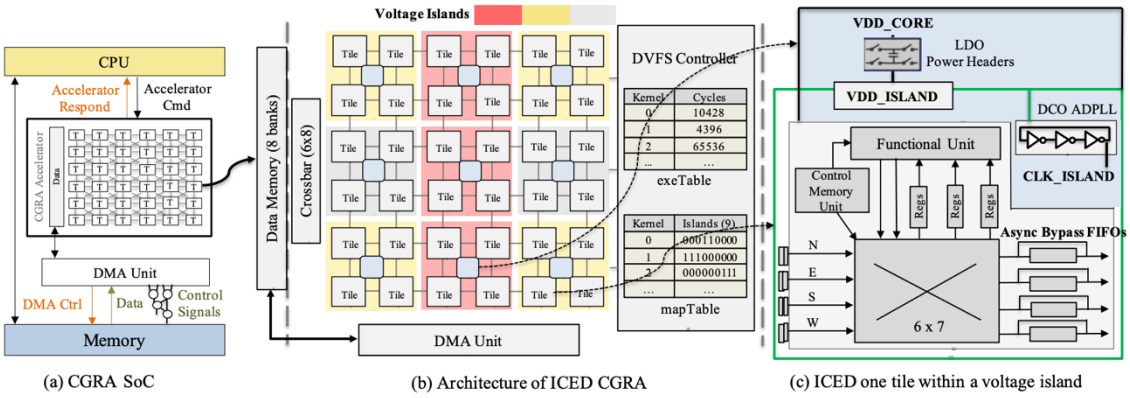


Fig. 5: ICED architecture – DVFS islands are highlighted in different colors. Each voltage island can be set to a voltage level at runtime. Red indicates `normal level`, yellow indicates `relax level`, and grey indicates `rest level`.

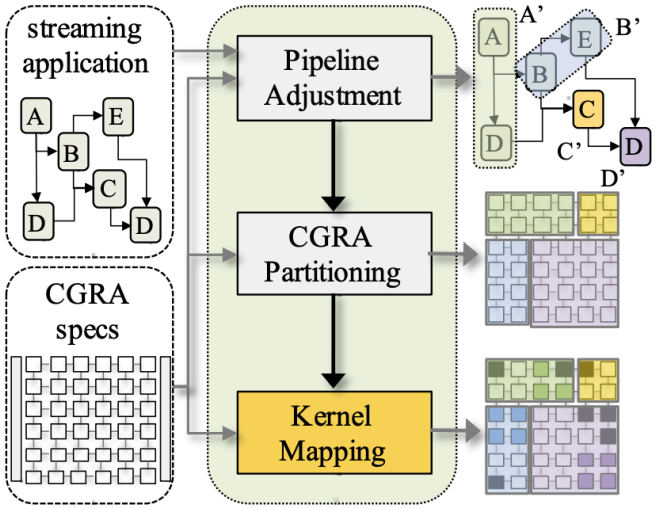


Fig. 7: ICED integrated compiler framework – The framework is implemented on top of the LLVM infrastructure [18]. Different colors in the bottom-right CGRA demonstrate that the islands allocated to each kernel are dynamically adjusted to different DVFS levels.

C.8) Pruning CGRA

In the paper "Enhancing CGRA Efficiency Through Aligned Compute and Communication Provisioning" [25], we focus on pruning CGRA communication to align with computation.

Coarse-grained Reconfigurable Arrays (CGRAs) are domainagnostic accelerators that enhance the energy efficiency of resource-constrained edge devices. The CGRA landscape is diverse, exhibiting trade-offs between performance, efficiency, and architectural specialization. However, CGRAs often overprovision communication resources relative to their modest computing capabilities. This occurs because the theoretically provisioned programmability for CGRAs often proves superfluous in practical implementations.
In this paper, we propose Plaid, a novel CGRA architecture and compiler that aligns compute and communication capabilities, thereby significantly improving energy and area efficiency while preserving its generality and performance. We demonstrate that the dataflow graph, representing the target application, can be decomposed into smaller, recurring

communication patterns called motifs. The primary contribution is the identification of these structural motifs within the dataflow graphs and the development of an efficient collective execution and routing strategy tailored to these motifs. The Plaid architecture employs a novel collective processing unit that can execute multiple operations of a motif and route related data dependencies together. The Plaid compiler can hierarchically map the dataflow graph and judiciously schedule the motifs. Our design achieves a 43% reduction in power consumption and 46% area savings compared to the baseline high-performance spatio-temporal CGRA, all while preserving its generality and performance levels. In comparison to the baseline energy-efficient spatial CGRA, Plaid offers a 1.4× performance improvement and a 48% area savings, with almost the same power.

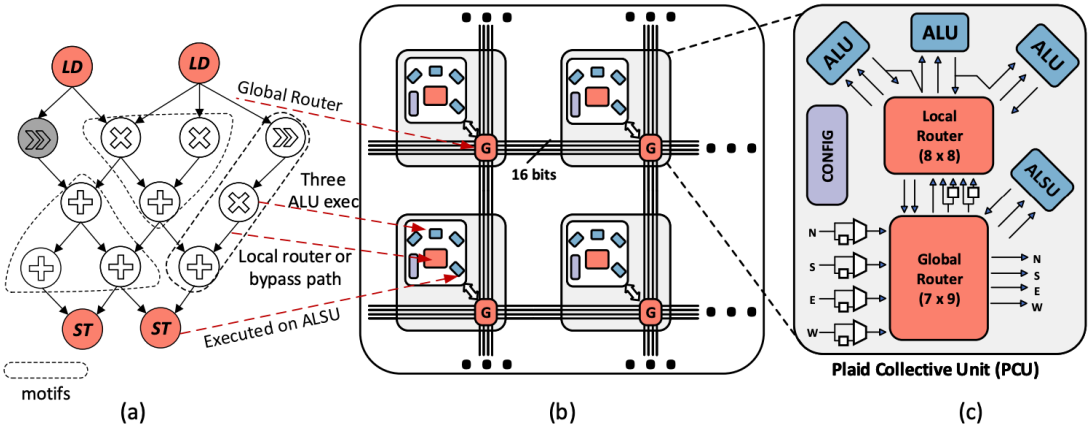


Figure 9. (a) Hierarchical DFG with multiple motifs. (b) Plaid architecture overview (c) Plaid Collective Unit

D) Application Optimizations

The resource-constrained edge devices like PACE impose innate restrictions on the computation achievable within power budget, even with innovations at hardware level. Application-level transformations that reduce the complexity of the computation are essential to accomplish more with the same compute capability. Deep learning is computationally and memory intensive. Therefore, we need to develop techniques to compress the deep neural networks without compromising their accuracy.

D.1) Data-Free Knowledge Distillation

Knowledge distillation is a compression approach that shifts knowledge from a large teacher model into a small student model by learning the class distributions output. Data-Free Knowledge Distillation (KD) allows knowledge transfer from a trained neural network (teacher) to a more compact one (student) in the absence of original training data. Existing works use a validation set to monitor the accuracy of the student over real data and report the highest performance throughout the entire process. However, validation data may not be available at distillation time either, making it infeasible to record the student snapshot that achieved the peak accuracy.

Therefore, a practical data-free KD method should be robust and ideally provide monotonically increasing student accuracy during distillation. This is challenging because the student experiences knowledge degradation due to the distribution shift of the synthetic data. A straightforward approach to overcome this issue is to store and rehearse the generated samples periodically, which increases the memory footprint and creates privacy concerns [4, 5].

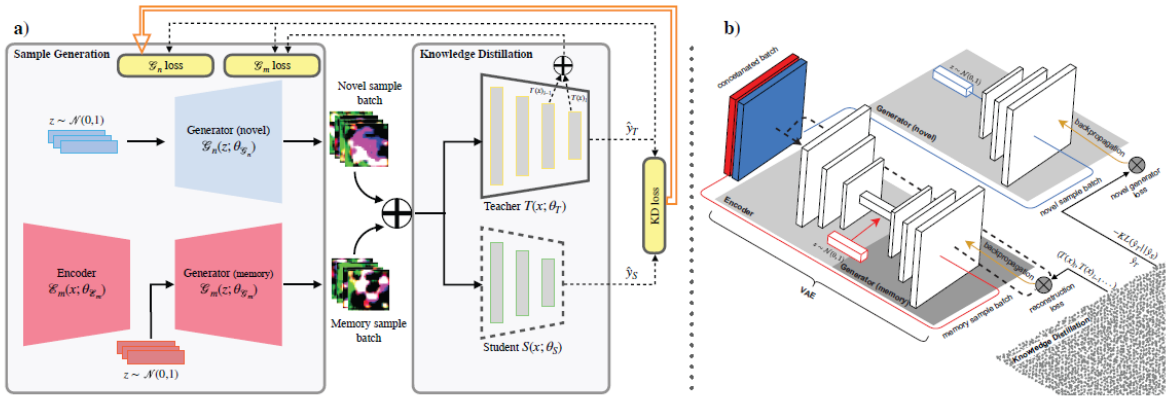


Figure 34: Our robust data-free knowledge-distillation approach

We propose to model the distribution of the previously observed synthetic samples with a generative network. In particular, we design a Variational Autoencoder (VAE) with a training objective that is customized to learn the synthetic data representations optimally. The student is rehearsed by the generative pseudo replay technique, with samples produced by the VAE [4, 5]. Hence knowledge degradation can be prevented without storing any samples (Figure 34)

Experiments on image classification benchmarks show that our method optimizes the expected value of the distilled model accuracy while eliminating the large memory overhead incurred by the sample-storing methods as shown in Figure 35 below.

	MNIST $\mathcal{T}$:LeNet5 (98.9%) $\mathcal{S}$:LeNet-half			CIFAR10 $\mathcal{T}$:ResNet34 (95.4%) $\mathcal{S}$:ResNet18			CIFAR100 $\mathcal{T}$:ResNet34 (77.9%) $\mathcal{S}$:ResNet18			Tiny ImageNet $\mathcal{T}$:ResNet34 (71.2%) $\mathcal{S}$:ResNet18		
Method	μ	σ^2	acc_{max}	μ	σ^2	acc_{max}	μ	σ^2	acc_{max}	μ	σ^2	acc_{max}
Train with data	98.7	0.5	98.9	89.0	8.1	95.2	71.3	8.1	77.1	60.2	8.8	64.9
DAFL	87.3	6.6	98.2	62.6	17.1	92.0	52.5	12.8	74.5	39.5	**10.3**	52.2
DFAD	63.5	6.8	98.3	86.1	12.3	93.3	54.9	12.9	67.7	-	-	-
CMI	memory: N.A.			memory: 250 MB			memory: 500MB			memory: 2.7 GB		
	-	-	-	82.4	16.6	**94.8**	55.2	24.1	77.0	-	-	-
MB-DFKD	memory: 6.7 MB			memory: 20 MB			memory: 20 MB			memory: 20MB		
	88.6	3.2	98.3	83.3	16.4	92.4	64.4	18.3	75.4	45.7	11.5	53.5
PRE-DFKD (ours)	memory: 2.1 MB			memory: 2.1 MB			memory: 2.1 MB			memory: 2.1 MB		
	90.3	**1.9**	**98.3**	**87.4**	**10.3**	94.1	**70.2**	**11.1**	**77.1**	**46.3**	11.0	**54.2**

Figure 35: Accuracy comparison of our data-free KD approach with state-of-the-art

The data-free distillation technique presented earlier is based on generating synthetic samples to substitute the real training set. To generate informative samples that are aligned with the real data distribution, the information encapsulated in the teacher model is leveraged, which is called model inversion. However, in the absence of data, available approaches often rely on naive assumptions about the properties a “realistic” sample should satisfy. Therefore, the generated samples can be sub-optimal for distillation, as visualized in Figure 36.

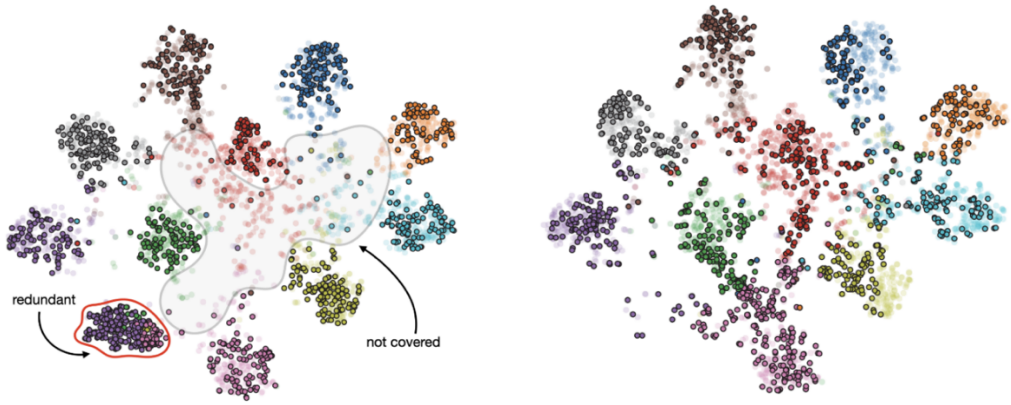


Figure 36: (Left) Synthetic feature distribution by a data-free baseline. (Right) Condensed data-guided synthetic feature distribution (ours).

We propose utilizing condensed samples as templates while generating synthetic samples via model inversion. Our method [16] involves first feeding the condensed samples to the teacher model to estimate the per-class feature distributions of the target data. Then, the feature distributions of the generated samples are aligned with the estimated ones. We achieve this by configuring a feature discriminator that competes against the sample generator to distinguish condensed samples from the generated ones. By doing so, the generator is forced to produce synthetic samples with semantic similarity to the condensed ones, thus reducing the risk of a domain gap and improving accuracy. One of the major advantages of our approach is its versatility, as it can be applied on top of any data-free KD method to significantly improve the accuracy of the student. This is evident from the experimental evaluation where we record improvement of up to 11.44% compared to different state-of-the-art KD baselines across multiple model pairs and datasets. Remarkably, even using as few as one condensed sample per class results in noticeable accuracy improvement.

D.2) Online Continual Learning

Once deployed on edge devices, a deep neural network model should dynamically adapt to newly discovered environments and personalize its utility for each user. The system must be capable of continual learning, i.e., learning new information from a temporal stream of data in situ without forgetting previously acquired knowledge. However, the prohibitive intricacies of such a personalized continual learning framework stand at odds with limited compute and storage on edge devices. Existing continual learning methods rely on massive memory storage to preserve the past data while learning from the incoming data stream. We propose Chameleon [7], a hardware-friendly continual learning framework for user-centric training with dual replay buffers. The proposed strategy leverages the hierarchical memory structure available on most edge devices, introducing a short-term replay store in the on-chip memory and a long-term replay store in the off-chip memory to acquire new information while retaining past knowledge (Figure 37). Extensive experiments on two largescale continual learning benchmarks demonstrate the efficacy of our proposed method, achieving better or comparable accuracy than existing state-of-the-art techniques while reducing the memory footprint by roughly 16× (Figure 38). Our method achieves up to 7× speedup and energy efficiency on edge devices such as ZCU102 FPGA, NVIDIA Jetson Nano and Google's EdgeTPU.

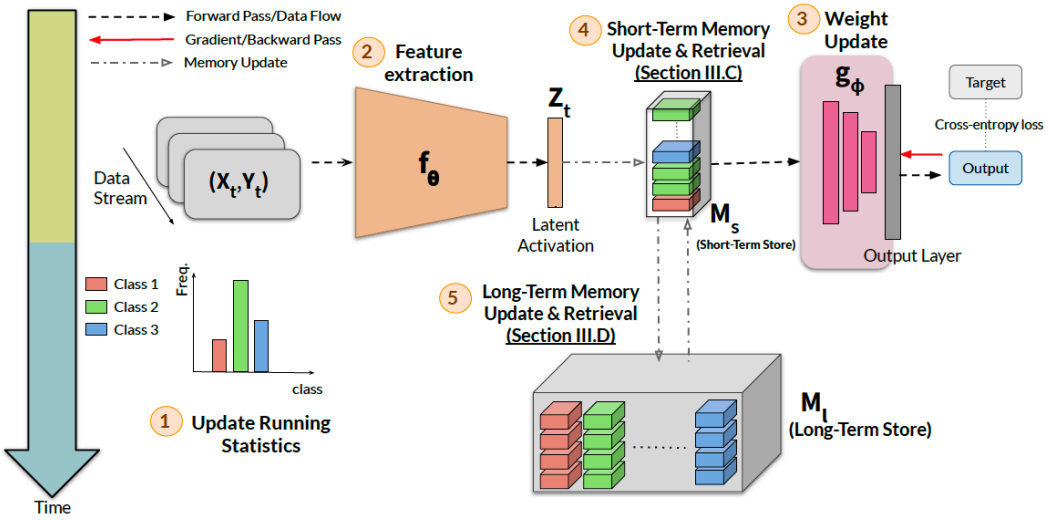


Figure 37: Schematic illustration of the proposed Chameleon framework for continual model personalization

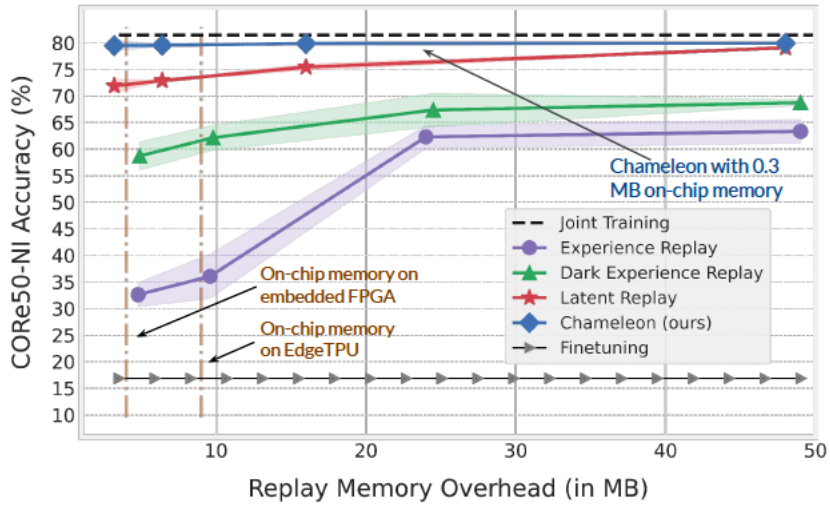


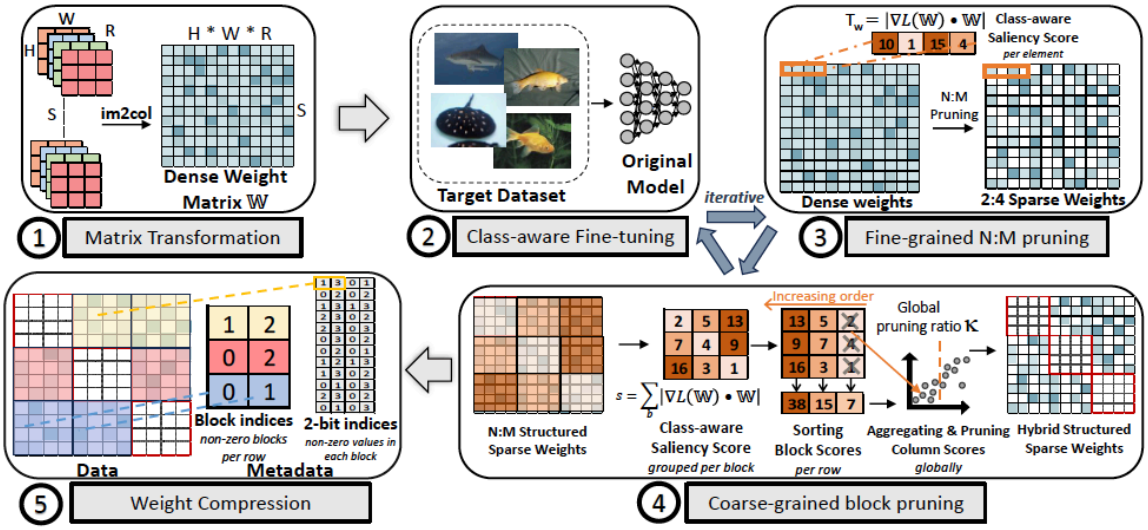
Figure 38: Accuracy comparison of different continual learning methods with varying memory budgets

D.3) Hybrid Structured Sparsity for Class-aware Model Pruning

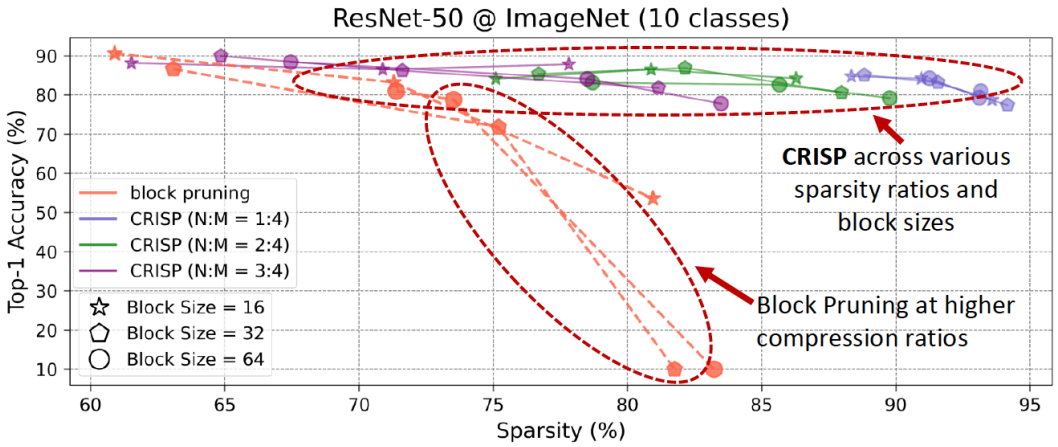
In our paper "CRISP: Hybrid Structured Sparsity for Class-aware Model Pruning," [17] we address the inefficiencies of universal machine learning models by tailoring them to focus on user-specific classes, which enhances computational efficiency. Traditional pruning methods, whether unstructured or structured, fall short in balancing hardware compatibility and model accuracy, especially at high sparsity levels.

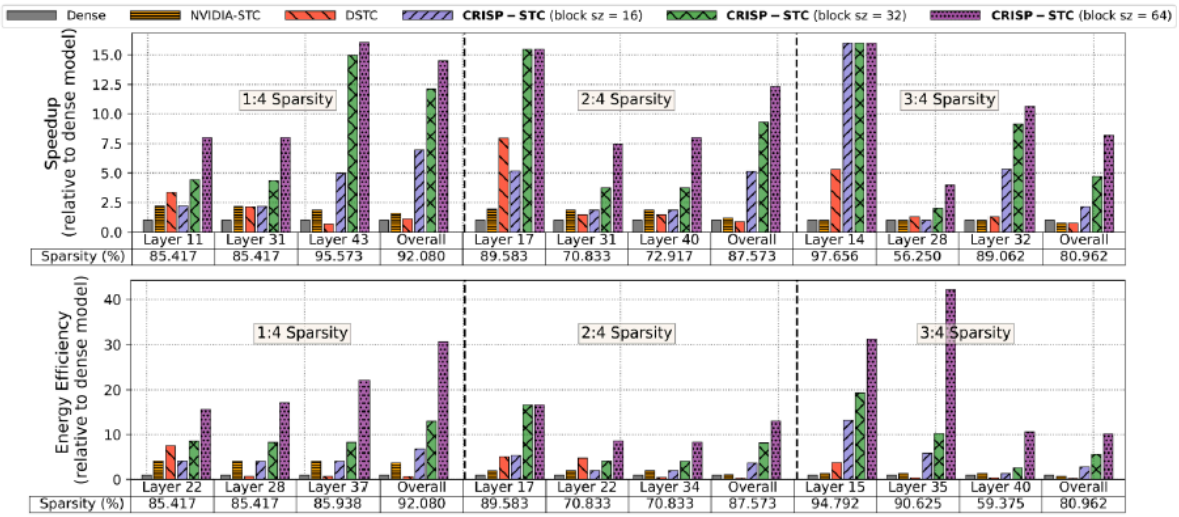
To tackle these challenges, we introduce CRISP, a novel pruning framework that leverages a hybrid structured sparsity pattern. Our method combines fine-grained N:M structured sparsity and coarse-grained block sparsity. This approach not only retains crucial weights for user-specific classes but also facilitates efficient hardware acceleration. CRISP employs a gradient-based class-aware saliency score to guide the pruning process, ensuring high accuracy while achieving significant memory and computational efficiency.

Our experiments on popular models such as ResNet-50, VGG-16, and MobileNetV2, using datasets like ImageNet and CIFAR-100, demonstrate that CRISP maintains comparable accuracy to dense models while reducing latency and energy consumption by up to 14× and 30×, respectively. By integrating these structured sparsity patterns and optimizing for hardware implementation, CRISP offers a robust solution for deploying efficient, personalized machine learning models on edge devices.



CRISP Framework





CRISP accuracy, speedup and energy efficiency

D.4) Lightweight robotic mapping for navigation

PACE 2.0 accelerator is expected to be used in robotic mapping for navigation. This involves reconstruction of complete scenes and requires the recovery of camera motions to align multiple images or depth scans. A two-step workflow is commonly used to recover the camera motions. In the first step, the relative motion between any two images or point clouds is recovered. The second step jointly optimizes the relative motions from the first step to recover the absolute camera poses, a problem known as Transformation Synchronization. Unfortunately, the pairwise registration process in the first step can be noisy and corrupted with outliers, leading to inaccurate or wrong input relative motions. This makes Transformation Synchronization extremely challenging.

Our work addresses the second problem in [14]. Our approach takes as input a set of noisy pairwise relative motions with outliers and outputs the absolute poses (Figure 39) and an inlier/outlier estimate of each relative pose. Specifically, we take inspiration from iterative optimization methods and propose to use a single recurrent GNN for the entire optimization. Our network takes in a viewgraph where the nodes represent the absolute poses that we wish to recover, and the edges correspond to the noisy input relative motions. We circumvent the difficulties in predicting absolute poses by using an iterative scheme where each iteration predicts incremental updates to the absolute poses. The increments are parameterized in the tangent space so that the predicted poses remain within the manifold. To reduce the influence of outliers, we predict attention weights using a subnetwork to weigh the individual messages before aggregation. Our resulting network does not contain multiple stages and is end-to-end trainable. It also does not require additional information from the pairwise matching and is thus compatible with any pairwise registration method or input data modality.

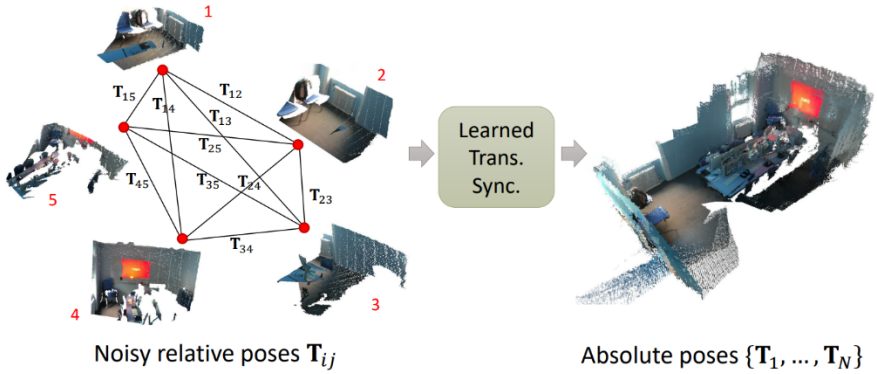


Figure 39: The goal of transformation synchronization is to take noisy relative poses with outliers (e.g., from point cloud pairwise registration) as input and recover the absolute poses.

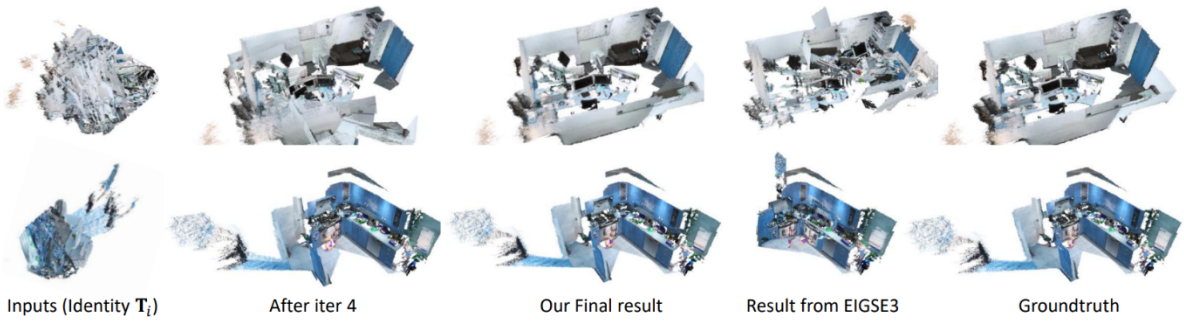


Figure 40: Qualitative rigid motion synchronization results on ScanNet (trained on ScanNet). Each view graph consists of 30-point cloud fragments.

We demonstrate the effectiveness of our method through experiments on synthetic and real-world rotation averaging datasets, as well as rigid motion synchronization on a real-world multi-viewpoint cloud registration dataset. Despite its simplicity, our network achieves competitive performance to existing works on the real-world rotation averaging dataset, and state-of-the-art performance for the remaining datasets. This is despite not employing any additional tricks, e.g., pre-filtering of the relative motions, or making use of additional information from pairwise matching or raw input data. Figure 40 shows a qualitative result on a real-world dataset compared with the ground truth.

E) Summary

We have achieved dual technical milestone for this programme. On one hand, we have developed the entire system stack (application, compiler, architecture, circuits optimization, and chip tapeout) for the PACE 1.0 accelerator. We are the first in the world to achieve this goal of end-to-end design covering application optimization to chip tapeput. On the other hand, we have made significant novel scientific contributions that will inform our design of the next-generation PACE accelerator.

References (Papers published, under submission or under preparation from the grant):

[1] PANORAMA: Divide-and-Conquer Approach for Mapping Complex Loop Kernels on CGRA
Dhananjaya Wijerathne, Zhaoying Li, Thilini Kaushalya Bandara, Tulika Mitra. 59th ACM/IEEE Design Automation Conference (DAC), 2022 **Publicity Paper** Artifact Link

[2] LISA: Graph Neural Network based Portable Mapping on Spatial Accelerators
Zhaoying Li, Dan Wu, Dhananjaya Wijerathne, Tulika Mitra. 28th IEEE International Symposium on High-Performance Computer Architecture (HPCA), 2022 Artifact Link **Distinguished Artifact Award**

[3] REVAMP: A Systematic Framework for Heterogeneous CGRA Realization.
Thilini Kaushalya Bandara, Dhananjaya Wijerathne, Tulika Mitra, Li-Shiuan Peh. 27th ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS), 2022 Artifact Link

[4] Robust and Resource-Efficient Data-Free Knowledge Distillation by Generative Pseudo Replay
Kuluhan Binici, Shivam Aggarwal, Nam Trung Pham, Karianto Leman, Tulika Mitra. 36th AAAI Conference on Artificial Intelligence (AAAI), 2022 Artifact Link

[5] Preventing Catastrophic Forgetting and Distribution Mismatch in Knowledge Distillation via Synthetic Data. Kuluhan Binici, Nam Trung Pham, Tulika Mitra. Winter Conference on Applications of Computer Vision (WACV), 2022

[6] An Energy-Efficient Processing Element Design for Coarse-Grained Reconfigurable Architecture on FPGA. Lingzhi Su, Wang Ling Goh, Jingjing Lan, Vishnu P. Nambiar, Anh Tuan Do, Dassanayake Mudiyanselage Thilini Kaushalya Bandara, Aditi Kulkarni Mohite, Wang Bo. 11th International Conference on Communications, Circuits and Systems (ICCCAS) 2022.

[7] Chameleon: Dual Memory Replay for Online Continual Learning on Edge Devices. Shivam Aggarwal, Kuluhan Binici, Tulika Mitra. Design Automation and Test in Europe (DATE), 2023
Artifact Link

[8] Power-Performance Characterization of TinyML Systems. Yujie Zhang, Dhananjaya Wijerathne, Zhaoying Li, Tulika Mitra. 40th IEEE International Conference on Computer Design (ICCD) 2022

[9] HiMap: Fast and Scalable High-Quality Mapping on CGRA via Hierarchical Abstraction
Dhananjaya Wijerathne, Zhaoying Li, Anuj Pathania, Tulika Mitra, Lothar Thiele
IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD), 41(10) 2022

[10] Coarse-Grained Reconfigurable Array (CGRA). Zhaoying Li, Dhananjaya Wijerathne, Tulika Mitra
Book chapter in "Handbook of Computer Architecture", Springer (Invited) 2023

[11] GraphWave: A Highly-Parallel Compute-at-Memory Graph Processing Accelerator. Jinho Lee, Burin Amornpaisannon, Tulika Mitra, Trevor E. Carlson. Design Automation and Test in Europe (DATE), 2022

[12] Morpher: An Open-Source Integrated Compilation and Simulation Framework for CGRA. Dhananjaya Wijerathne, Zhaoying Li, Manupa Karunaratne, Li-Shiuan Peh, Tulika Mitra. Workshop on Open-Source EDA Technology at 41st ACM/IEEE International Conference on Computer Aided Design (ICCAD), November 2022. Artifact Link

[13] Pipelined CNN Inference on Heterogeneous Multi-Processor System-on-Chip. Ehsan Aghapour, Yujie Zhang, Anuj Pathania, Tulika Mitra. Book chapter in "Embedded Machine Learning for Cyber Physical, IoT, and Edge Computing", Springer (Invited) 2023

[14] Learning Iterative Robust Transformation Synchronization. Zi Jian Yew, Gim Hee Lee. International Conference on 3D Vision (3DV), 2021

[15] ASCENT: Communication Scheduling for SDF on Bufferless Software-defined NoC. Vanchinathan Venkataramani, Bruno Bodin, Aditi Kulkarni, Tulika Mitra, Li-Shiuan Peh. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD) 41(10) 2022

[16] Condensed Data-Guided Model Inversion for Knowledge Distillation. Kuluhan Binici, Shivam Aggarwal, Cihan Acar, Nam Trung Pham, Karianto Leman, Gim Hee Lee, Tulika Mitra. *Under submission.*

[17] CRISP: Dual Structured Sparsity for Model Personalization. Shivam Aggarwal, Kuluhan Binici, Tulika Mitra.
Shivam Aggarwal, Kuluhan Binici, Tulika Mitra. Design Automation and Test in Europe, 2024

[18] High-Performance Multi-Branch DNN Model Inference on Heterogeneous SoC. Yujie Zhang, Huiying Lan, Zhiyuan Ning, Peng Zan, Weidong Shao, Tulika Mitra. 42nd ACM/IEEE International Conference on Computer Aided Design (ICCAD) 2023. *Under review*

[19] FLEX : Introducing FLEXible Execution on CGRA with Spatio-Temporal Vector Dataflow. Thilini Kaushalya Bandara, Dan Wu, Tulika Mitra, Li-Shiuan Peh. 42nd ACM/IEEE International Conference on Computer Aided Design (ICCAD) 2023.

[20] Flip: Data-Centric Edge CGRA Accelerator. Dan Yu, Thilini Kaushalya Bandara, Zhaoying Li, Tulika Mitra. Dan Wu, Peng Chen, Thilini Kaushalya Bandara, Zhaoying Li, Tulika Mitra. ACM Transactions on Design Automation of Electronic Systems. 29(1) January 2024

[21] ASADI: Accelerating Sparse Attention using Diagonal-based In-situ Computing. Huize Li, Zhaoying Li, Zhenyu Bai, Tulika Mitra. 30th IEEE International Symposium on High-Performance Computer Architecture, 2024

[22] AM-CGRA: Reconfigurable Architecture for Irregular Applications. Rohan Juneja, Tulika Mitra, Li-Shiuan Peh. *Under review*

[23] Low-precision Post-Training Quantization with Custom Minifloats. Shivam Aggarwal, Alessandro Pappalardo, Michaela Blott, Tulika Mitra. FPL 2024

[24] SWAT: Scalable and EfficientWindow Attention-based dcstmTransformers Acceleration on FPGAs. Zhenyu Bai, Pranav Dangi, Huize Li, Tulika Mitra. 61st ACM/IEEE Design Automation Conference, 2024

[25] Enhancing CGRA Efficiency Through Aligned Compute and Communication Provisioning
Zhaoying Li, Pranav Dangi, Chenyang Yin, Thilini Kaushalya Bandara, Rohan Juneja, Cheng Tan, Zhenyu Bai, Tulika Mitra
30th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, 2025

[26] ICED: An Integrated CGRA Framework Enabling DVFS-Aware Acceleration
Cheng Tan, Miaomiao Jiang, Deepak Patil, Yanghui Ou, Zhaoying Li, Lei Ju, Tulika Mitra, Hyunchul Park, Antonino Tumeo, Jeff Zhang
57th IEEE/ACM International Symposium on Microarchitecture, 2024

[27] ZeD: A Generalized Accelerator for Variably Sparse Matrix Computations in ML
Pranav Dangi, Zhenyu Bai, Rohan Juneja, Dhananjaya Wijerathne, Tulika Mitra
33rd IEEE/ACM International Conference on Parallel Architectures and Compilation Techniques

[28] PACE: A Scalable and Energy Efficient CGRA in a RISC-V SoC for Edge Computing Applications
Vishnu Nambiar, Yi Sheng Chong, Thilini Bandara, Dhananjaya Wijerathne, Zhaoying Li, Rohan Juneja, Li-Shiuan Peh, Tulika Mitra, Anh Tuan Do
Hot Chips 2024

[29] A 360 GOPS/W CGRA in a RISC-V SoC with Multi-Hop Routers and Idle-State Instructions for Edge Computing Applications. Vishnu Nambiar, Yi Sheng Chong, Thilini Bandara, Dhananjaya Wijerathne, Zhaoying Li, Rohan Juneja, Li-Shiuan Peh, Tulika Mitra, Anh Tuan Do.
21st International SoC Design Conference (ISOCC)

[30] Sustainable Hardware Specialization
Pranav Dangi, Thilini Kaushalya Bandara, Saeideh Sheikhpour, Tulika Mitra, Lieven Eeckhout
43rd ACM/IEEE International Conference on Computer Aided Design, 2024