



TRISTAN has received funding from the Key Digital Technologies Joint Undertaking (KDT JU) under grant agreement nr. 101095947. The KDT JU receives support from the European Union's Horizon Europe's research and innovation programmes and participating states are Austria, Belgium, Bulgaria, Croatia, Cyprus, Czechia, Germany, Denmark, Estonia, Greece, Spain, Finland, France, Hungary, Ireland, Israel, Iceland, Italy, Lithuania, Luxembourg, Latvia, Malta, Netherlands, Norway, Poland, Portugal, Romania, Sweden, Slovenia, Slovakia, Turkey

# APROPOS Winter School

IBM Research Zurich, February 14-16, 2023

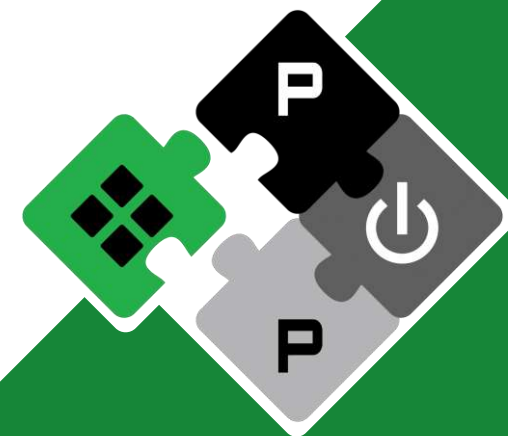
## Transprecision Floating-Point Units

Luca Bertaccini

IIS, ETH Zurich, Switzerland,

**PULP Platform**

Open Source Hardware, the way it should be!

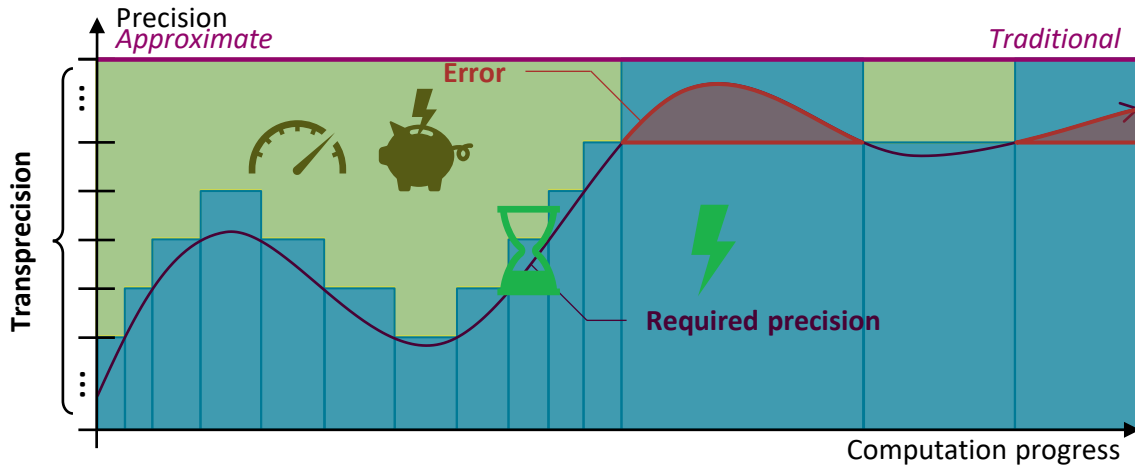


@pulp\_platform 

pulp-platform.org 

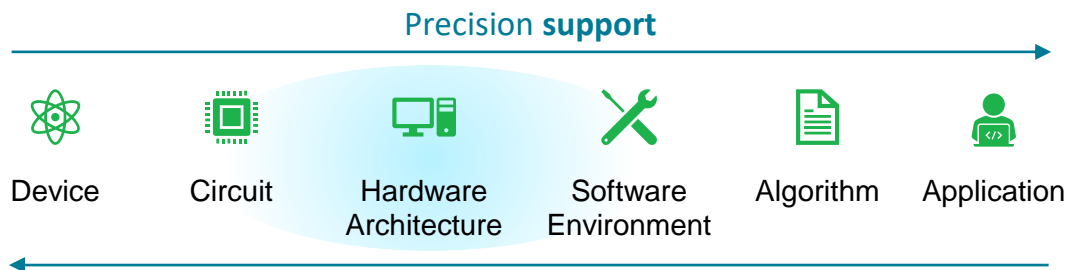
youtube.com/pulp\_platform 

# Transprecision Computing: More than just approximation



- Floating-Point is everywhere
- Conservative with precision
  - Use largest precision **everywhere** & **always**
- Approximate Computing?
- Transprecision Computing!
  - Just right precision **anywhere** & **anytime**
  - Potential energy savings & speedup

- Holistic approach necessary



Enable energy-proportional transprecision computing in hardware!

# Low-Precision Formats: Hardware Benefits



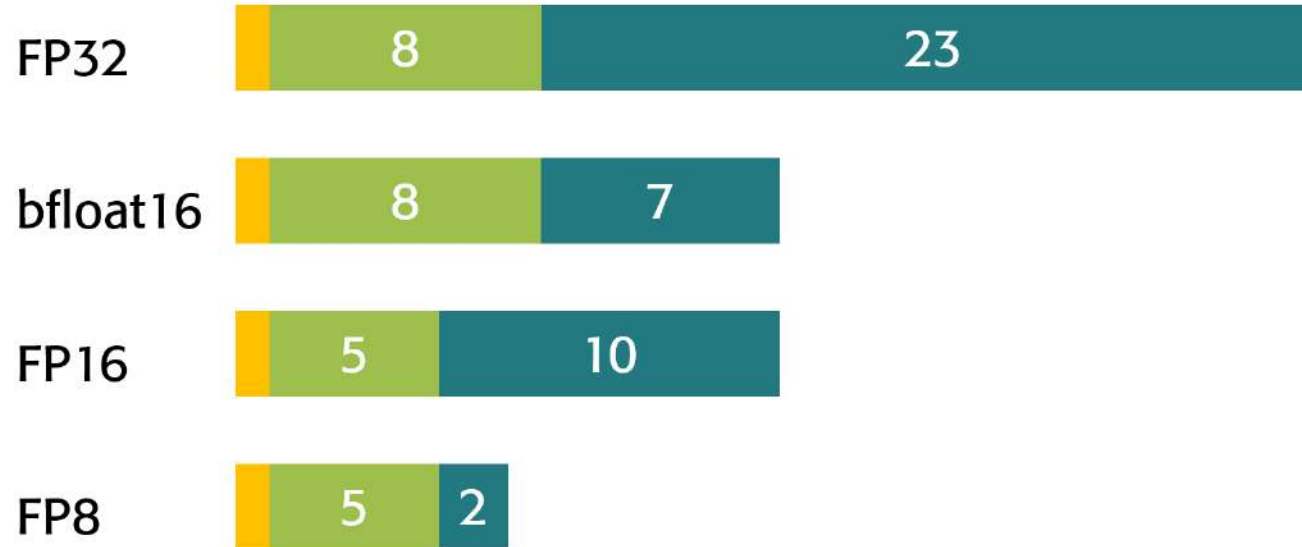
FP32



$$(-1)^s \times 2^{(\text{exp}) - \text{BIAS}} \times 1.(\text{mant})$$

$$\text{BIAS} = 2^{e-1} - 1$$

# Low-Precision Formats: Hardware Benefits

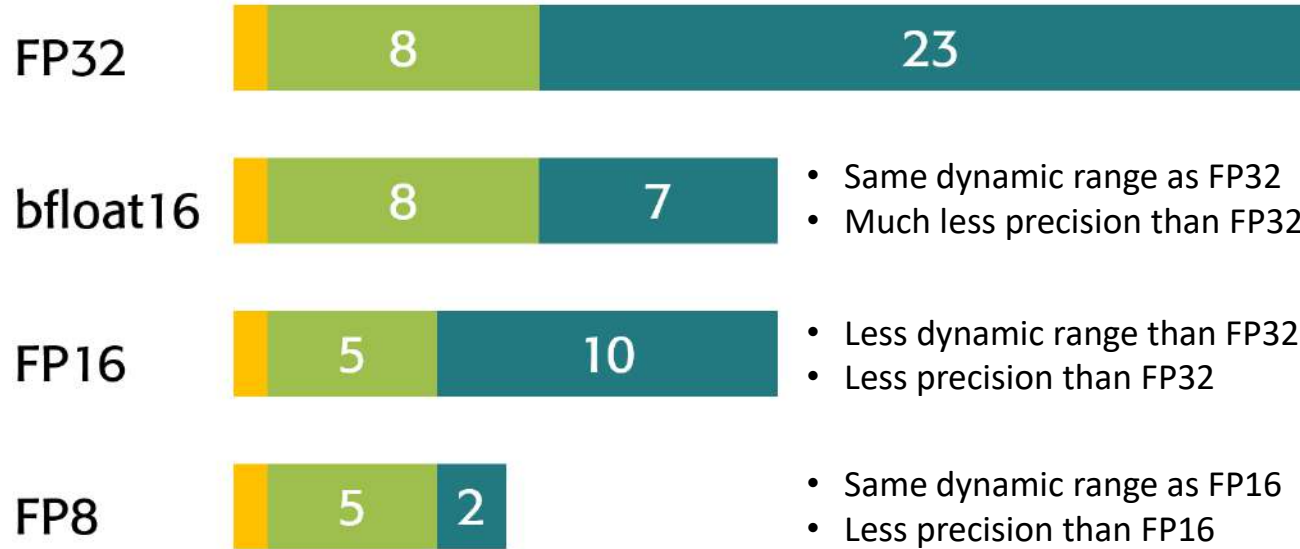


$$(-1)^s \times 2^{(\text{exp}) - \text{BIAS}} \times 1.(\text{mant})$$

$$\text{BIAS} = 2^{e-1} - 1$$

- Precision Tuning
  - “smallFloat” formats

# Low-Precision Formats: Hardware Benefits

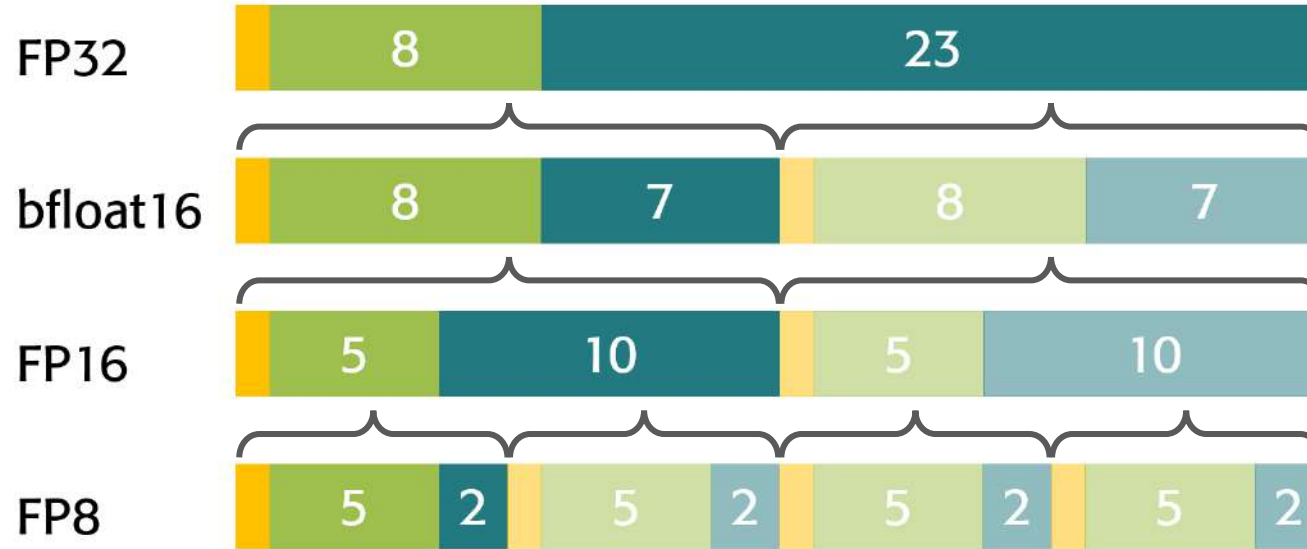


$$(-1)^s \times 2^{(\text{exp}) - \text{BIAS}} \times 1.(\text{mant})$$

$$\text{BIAS} = 2^{e-1} - 1$$

Format	# Representable values	Minimum Value (subnormal)	Minimum Value (normal)	Maximum Value
<b>FP32</b> (1,8,23)	$4.29 \times 10^9$	$\approx 1.40 \times 10^{-45}$	$\approx 1.18 \times 10^{-38}$	$\approx 3.40 \times 10^{38}$
<b>Bfloat16</b> (1,8,7)	65536	$\approx 9.2 \times 10^{-41}$	$\approx 1.18 \times 10^{-38}$	$\approx 3.40 \times 10^{38}$
<b>FP16</b> (1,5,10)	65536	$\approx 5.96 \times 10^{-8}$	$\approx 6.10 \times 10^{-5}$	$\approx 65504$
<b>FP8</b> (1,5,2)	256	$\approx 1.53 \times 10^{-5}$	$\approx 6.10 \times 10^{-5}$	$\approx 57344$

# Low-Precision Formats: Hardware Benefits



$$(-1)^s \times 2^{(\text{exp}) - \text{BIAS}} \times 1.(\text{mant})$$

$$\text{BIAS} = 2^{e-1} - 1$$

- Precision Tuning
  - “smallFloat” formats
  - SIMD vectors
  - Native hardware support

- Several benefits:
  - Higher Performance
  - Higher energy efficiency
  - Lower memory footprint

# Boosting the Energy Efficiency through ISA extension



- **General purpose:** tune precision & performance for **high energy-efficiency**
- General-purpose **RISC-V** CPUs can be extended with domain-specific ISA extension:
  - Opportunities for deeply optimizing the efficiency of the cores
  - Without compromising the baseline standardized ISA (still general-purpose)





# A Transprecision FPU

General purpose: tune precision & performance for  
high energy-efficiency



# An Open-Source Transprecision FPU



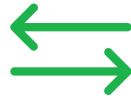
What is needed?



General-Purpose



Efficient



Flexible



Standard tools



Open source

- Support **standard** IEEE 754 FP
- Optimize **performance** and **energy-efficiency**
  - SIMD vectors
  - Special operations
- Many **target domains**
  - Operations & Formats
  - Technology
  - Architecture
- **Enable** new research

# CVFPU: Architecture



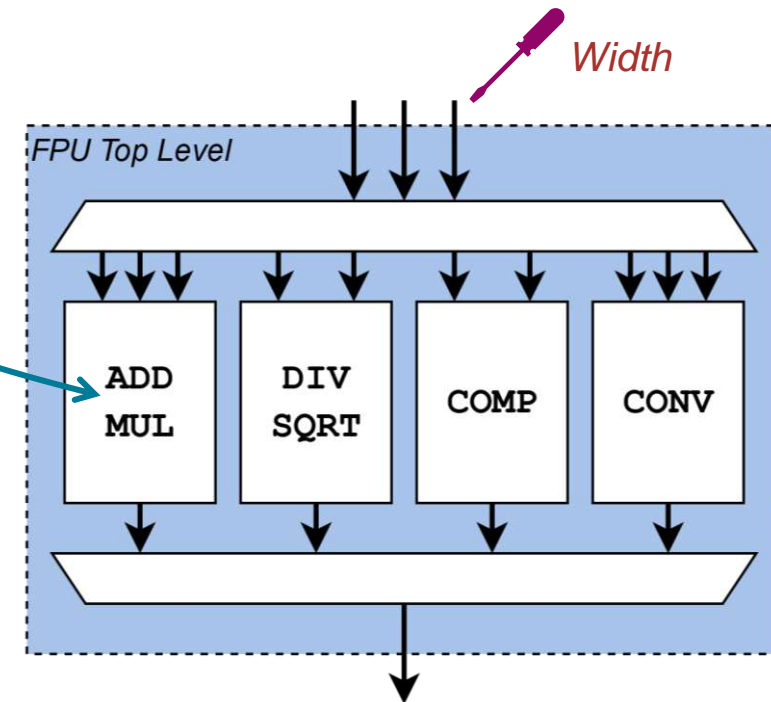
- Hierarchical approach

1. Grouped by **operation**

- (i) ADDitions and MULtiplications carried out on an FMA unit ( $FMA = a * b + c$ )
- (ii) division and square root
- (iii) comparisons
- (iv) conversions (FP $\leftrightarrow$ FP or INT $\leftrightarrow$ FP)



- Datapath width



# CVFPU: Architecture

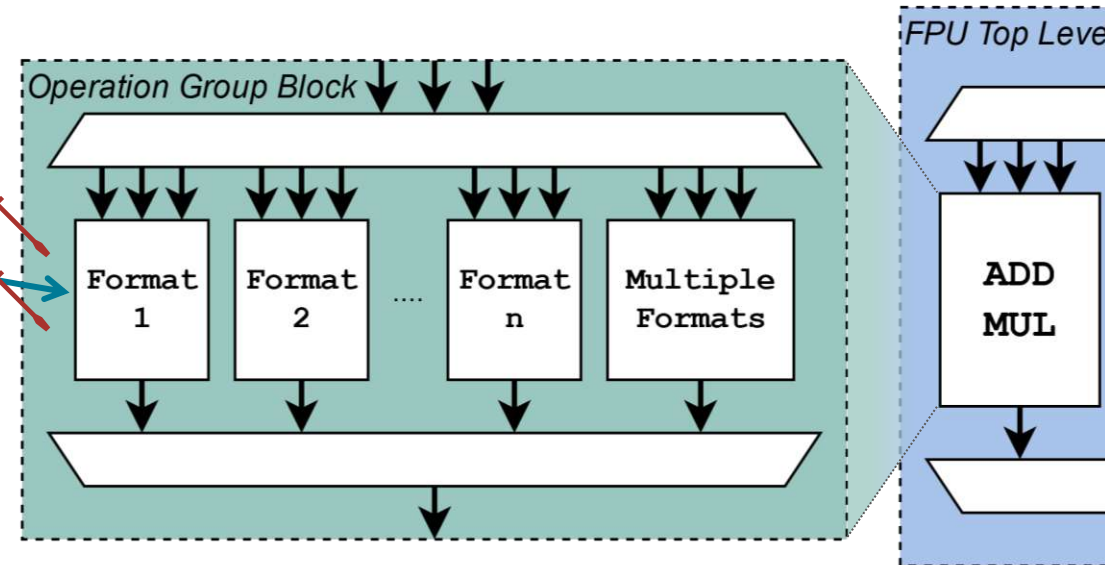


- Hierarchical approach:
  1. Grouped by **operation**
  2. Sliced by **format**



- Datapath width
- **Any formats** (new formats can be defined in a package at design time)
- **Stand-alone** or **multi-format** or **none**

*Formats*  
*Implementation*



# CVFPU: Architecture

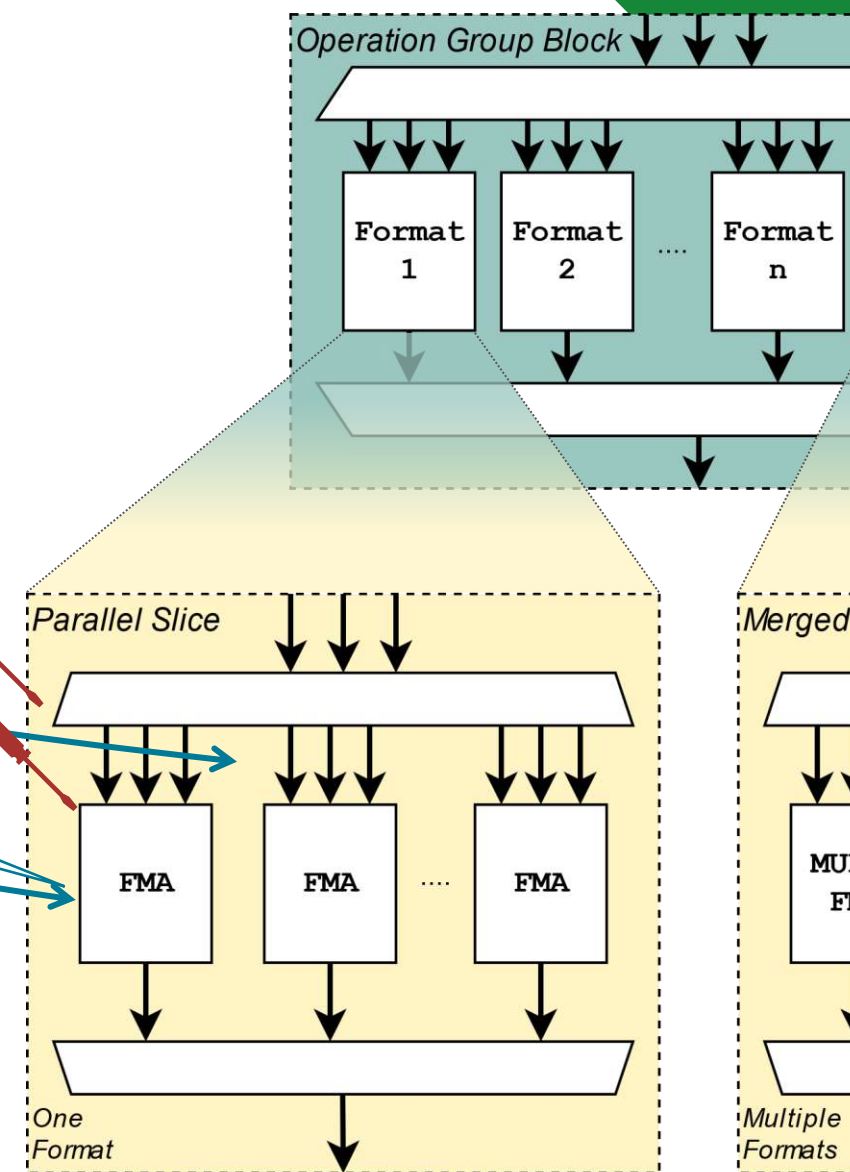
- Hierarchical approach:
  1. Grouped by **operation**
  2. Sliced by **format**
  3. Lanes for **SIMD**
  4. Execution unit

- FMA (Single-path architecture)
- Comparisons

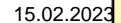


- Datapath width
- Any formats
- Stand-alone or multi-format or none
- SIMD support or **scalar only**
- **Pipelining** for execution unit

Vectors  
Pipelining



- Datapath width
- Any formats
- Stand-alone or multi-format or none
- SIMD support or scalar only
- Pipelining for execution unit



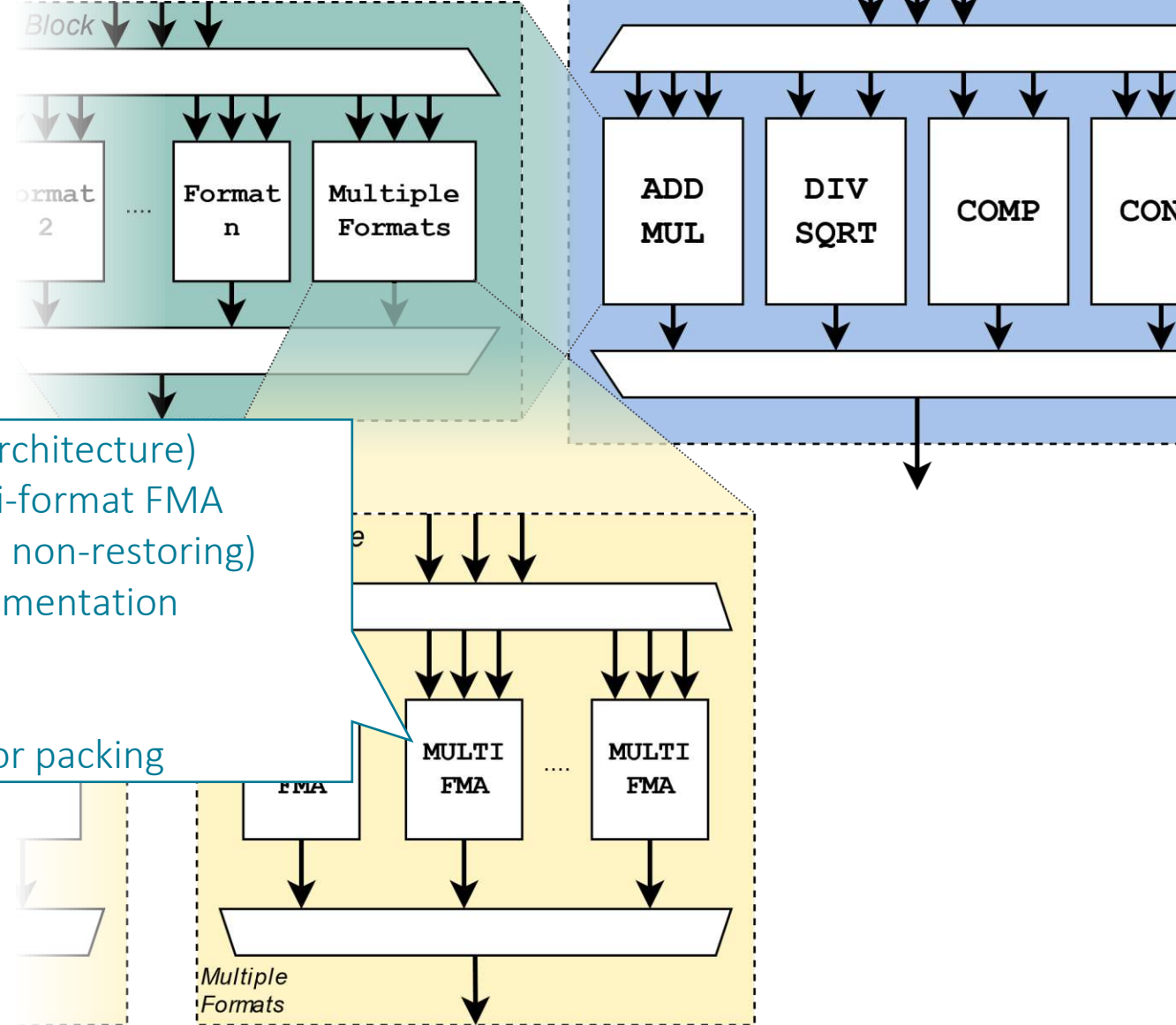
# CVFPU: Architecture

- Hierarchical approach:
  1. Grouped by **operation**
  2. Sliced by **format**
  3. Lanes for SIMD
  4. Execution unit



- Datapath width
- Any formats
- Stand-alone or multi-format or none
- SIMD support or scalar only
- Pipelining for execution unit

- **FMA** (Single-path architecture)
  - Supports multi-format FMA
- **DIV/SQRT** (Iterative non-restoring)
  - Blocking implementation
- **Casts**
  - Vector casts
  - Scalar → vector packing

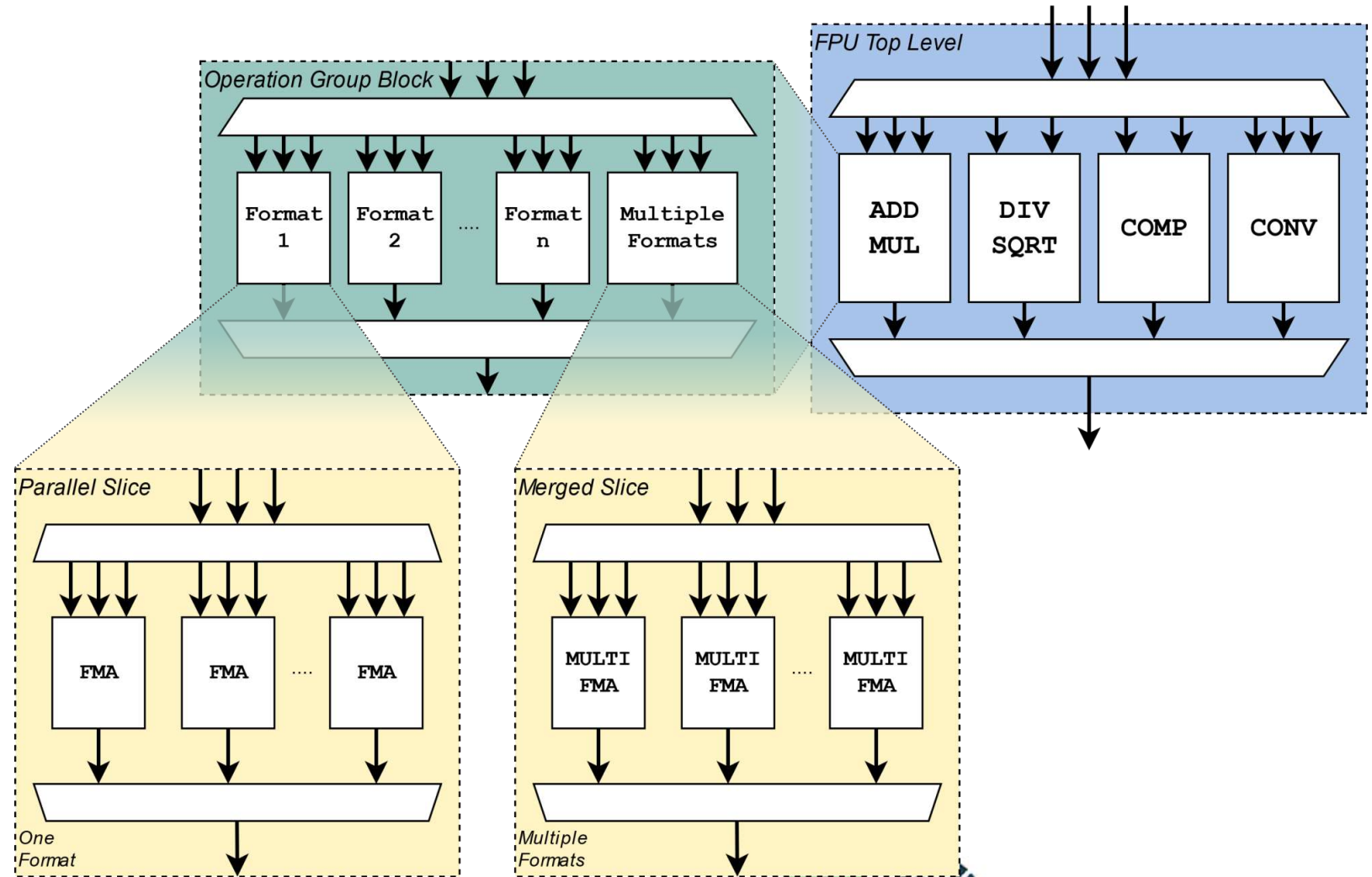




- Width
- Any formats
- Implementation
- SIMD
- Pipelining
- Open-Source
- Synthesizable
- Hackable



- Open-Source
- Synthesizable
- Hackable

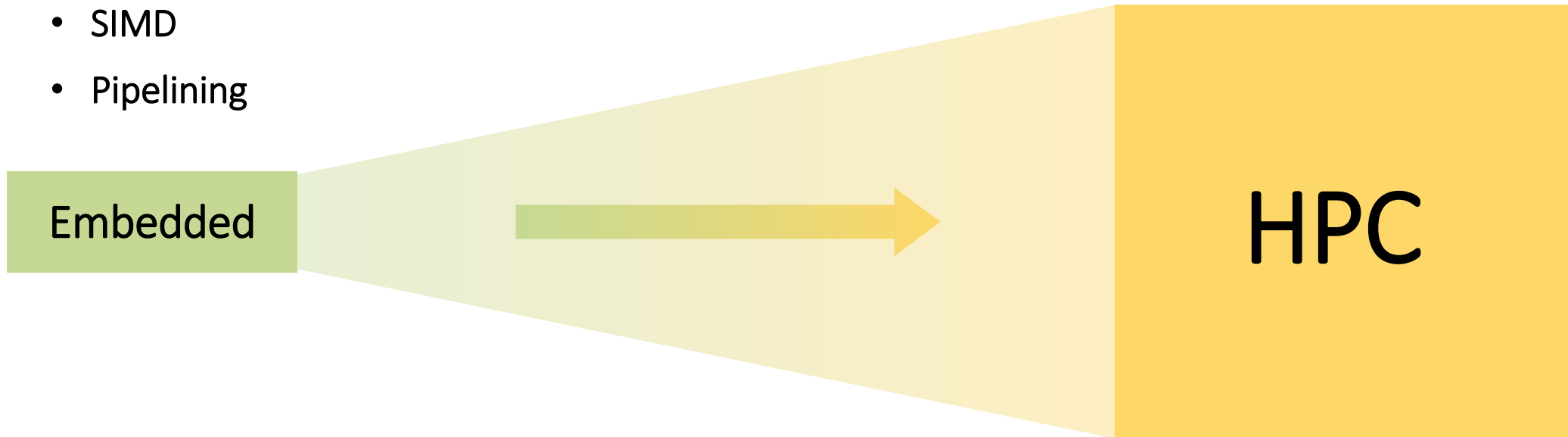


# CVFPU: Highly Configurable



CVFPU is highly parametrized to fit into a large variety of architectures:

- Width
- Formats
- Implementation
- SIMD
- Pipelining





# Programmability



- RISC-V **extensions** and **compiler** support
- Filter on FP16 data
- Native FP16
  - Error accumulation! Degraded by **14.6x**
- Native FP32
  - **Decrease** rel. error
  - **58%** more system energy
- Expanding FMA
  - Same precision as FP32
  - Comparable system energy to FP16

```
float16 *a, *b;
float16 sum = 0;
for (int i = 0; i < n; ++i)
    sum += a[i] * b[i];
```

```
float16 *a, *b;
float sum = 0;
for (int i = 0; i < n; ++i)
    sum += a[i] * b[i];
```

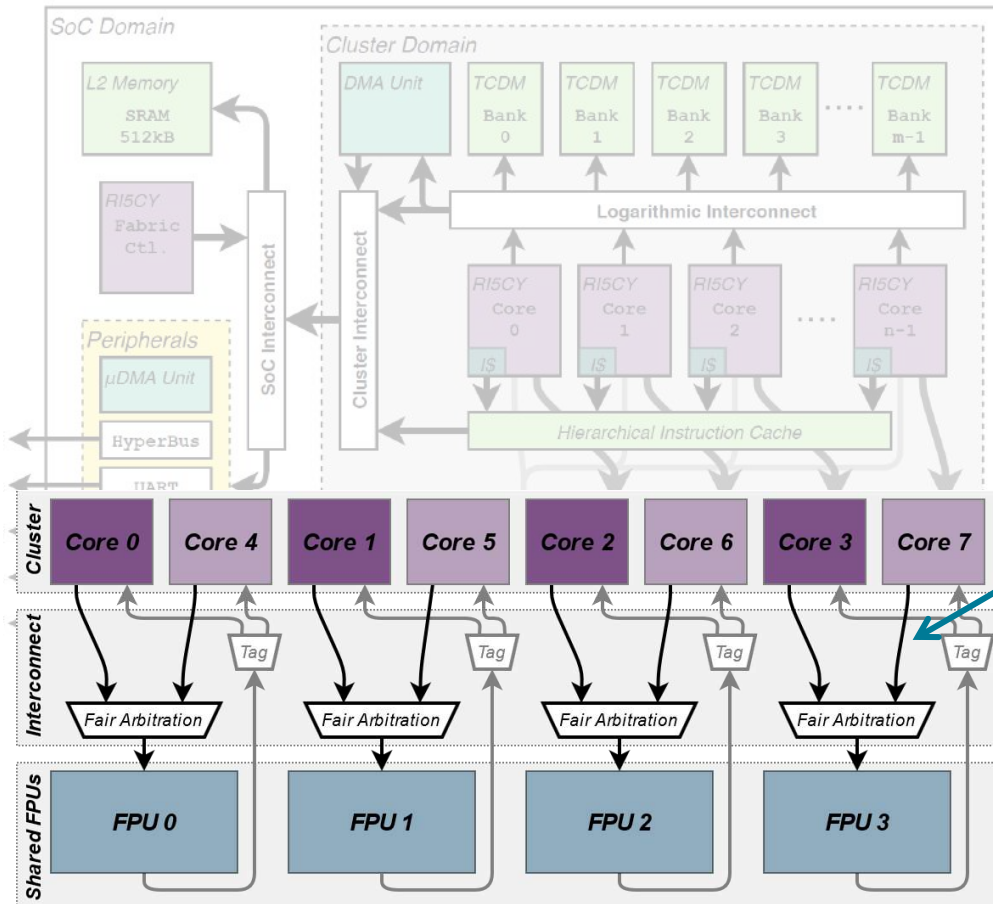
```
float16 *a, *b;
```

Load	lh	a4, 0(t0)
	lh	a5, 2(t0)
	lh	a6, 0(t1)
	lh	a7, 2(t1)
-----		
FP16	fmadd.h	a0, a4, a6, a0
	fmadd.h	a0, a5, a7, a0
-----		
Load	lh	a4, 0(t0)
	lh	a5, 2(t0)
	lh	a6, 0(t1)
	lh	a7, 2(t1)
-----		
FP16 to FP32	fcvt.s.h	a4, a4
	fcvt.s.h	a5, a5
	fcvt.s.h	a6, a6
	fcvt.s.h	a7, a7
-----		
FP32	fmadd.s	a0, a4, a6, a0
	fmadd.s	a0, a5, a7, a0
-----		
Load	lh	a4, 0(t0)
	lh	a5, 2(t0)
	lh	a6, 0(t1)
	lh	a7, 2(t1)
-----		
FP32	fcvt.s.h	a0, a4, a6
	fcvt.s.h	a0, a5, a7

FPnew is the perfect enabler for transprecision computing systems.  
Let's build some!

- Compiler int

# Embedded Transprecision Multi-Core Cluster

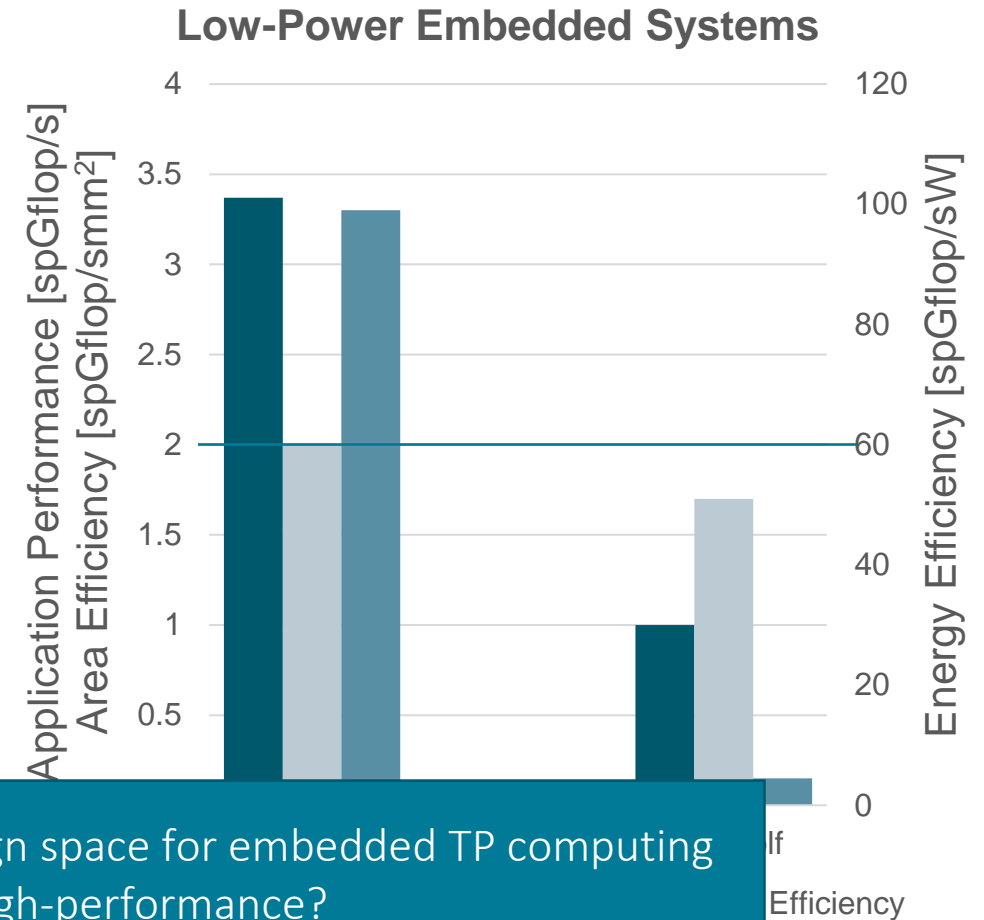


- Edge Computing for signal processing
- Build a **multi-core** cluster of RI5CY cores
  - FP32, FP16, bfloat16
  - **Share** TP-FPUs
- Design Space **Exploration**
  - # Cores
  - FPU/Core Ratio
  - # Pipeline stages
- **Implemented** in 22FDX
  - Post-layout power
- **Implemented** on FPGA
  - Benchmarks

# Multi-Core Cluster: Selected Results

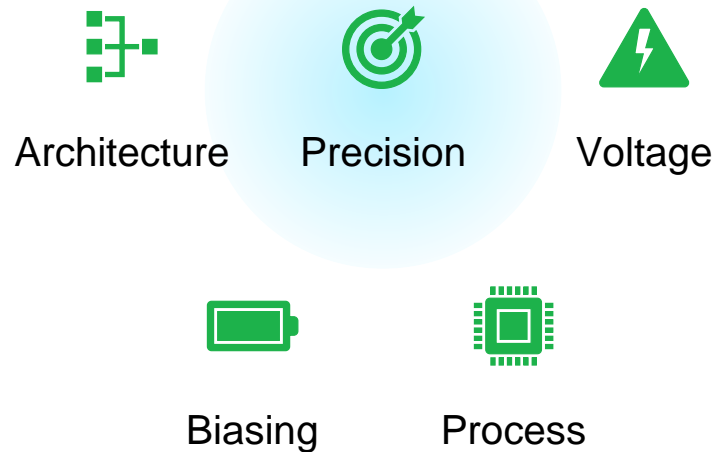
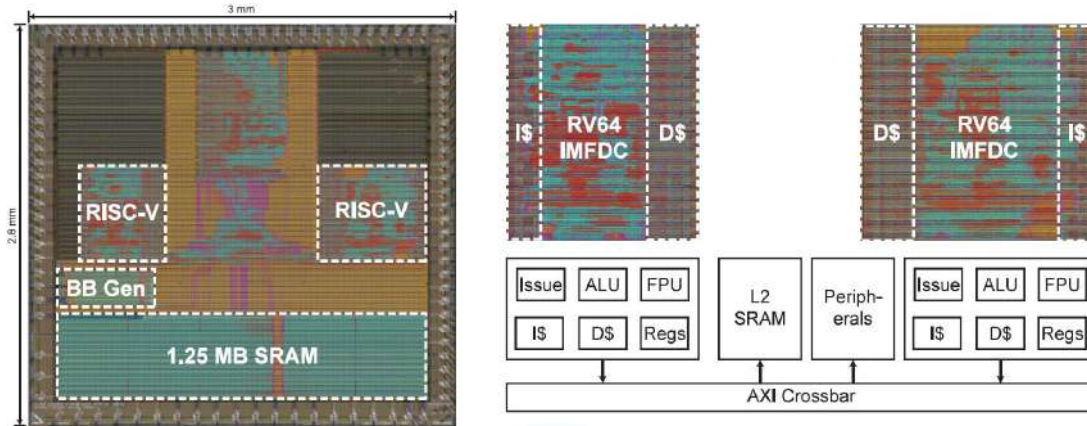


- 8x 4 Benchmarks, 18 architectures
- Best architectures
  - Performance: **3.4 spGflop/s**
  - Energy Efficiency: **99 spGflop/sW**
  - Area Efficiency: **2 spGflop/smm<sup>2</sup>**
- Compared to SoA:
  - **Outperforming** SoA low-power embedded systems

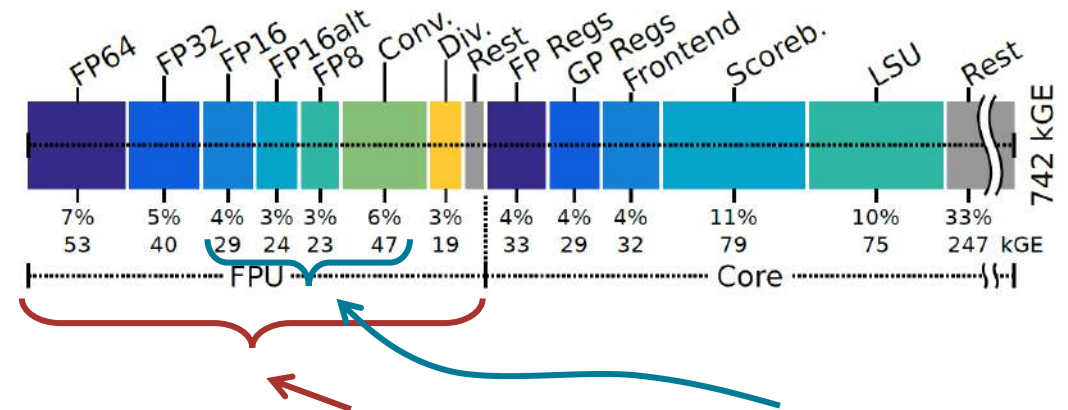


Our TP-FPU architecture opens a huge design space for embedded TP computing systems. What about high-performance?

# Kosmodrom: Application-class transprecision computing

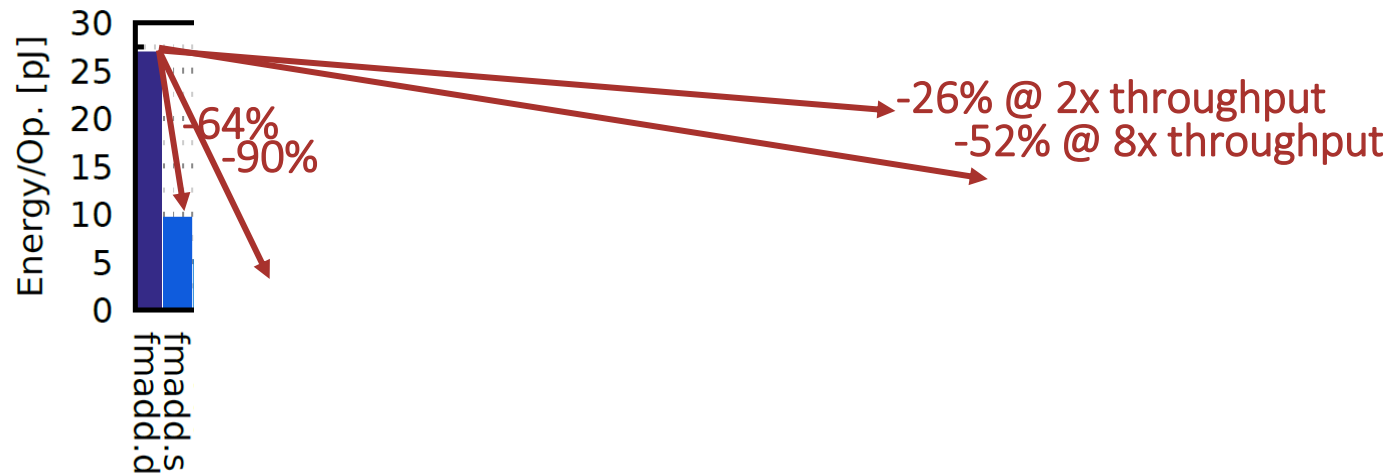


- Dual-core Ariane **ASIC** in 22nm FDX
- **Ariane**: 64-bit, 6-stage RISC-V with **CVFPU**
  - FP64, FP32, FP16, BF16, FP8
  - **SIMD** for FP32 – FP8



- TP-FPU is ~30% of the core, only +9% area

# Kosmodrom: TP-FPU Energy Measurements (in Silicon)



- Scalar: **equal** performance @ **reduced** energy
  - FP64: 13.4 pJ/flop
  - FP8: 1.27 pJ/flop

↓ 10.5x more efficient
- Superlinear scaling

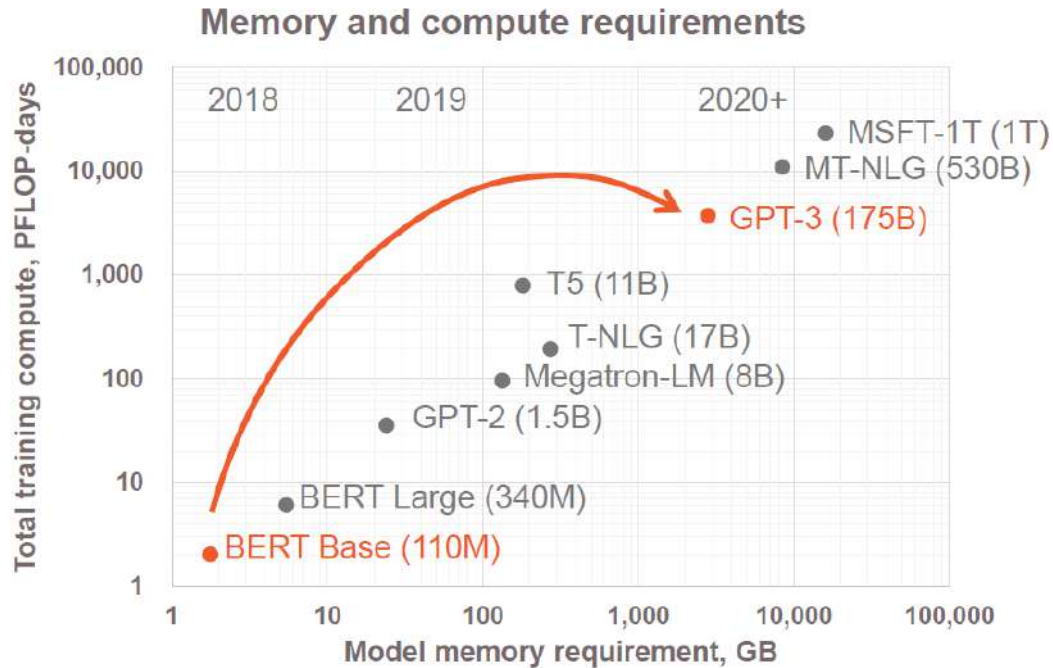
- Vector: **improved** performance @ ~~equal energy~~
  - FP64: 13.4 pJ/flop
  - FP8: 0.80 pJ/flop

↓ 16.8x more efficient
- Superlinear scaling
- Improved performance at improved energy



# A Transprecision FPU enhanced for NN training

# The Rapid Growth of AI



- NN models' memory and compute requirements are growing **rapidly**
- Technology scaling is not sufficient



- Required **algorithmic** and **architectural** advancements

*S. Lie, "Thinking outside the die: Architecting the ML accelerator of the future"*



# Algorithmic Advancements

- New low-precision data types:
  - 32-bit → 19-bit → 16-bit → 8-bit floating-point (FP) data types



- Lower memory requirements
- Opportunities for more efficient hardware architectures
- Wide interest for standardization (RISC-V FP SIG, IEEE P3109)



- New **mixed** and **low-precision training** algorithms have been developed to exploit the resilience of NN models to noise
- Expanding/Widening **operations** in which the accumulation is performed in higher precision

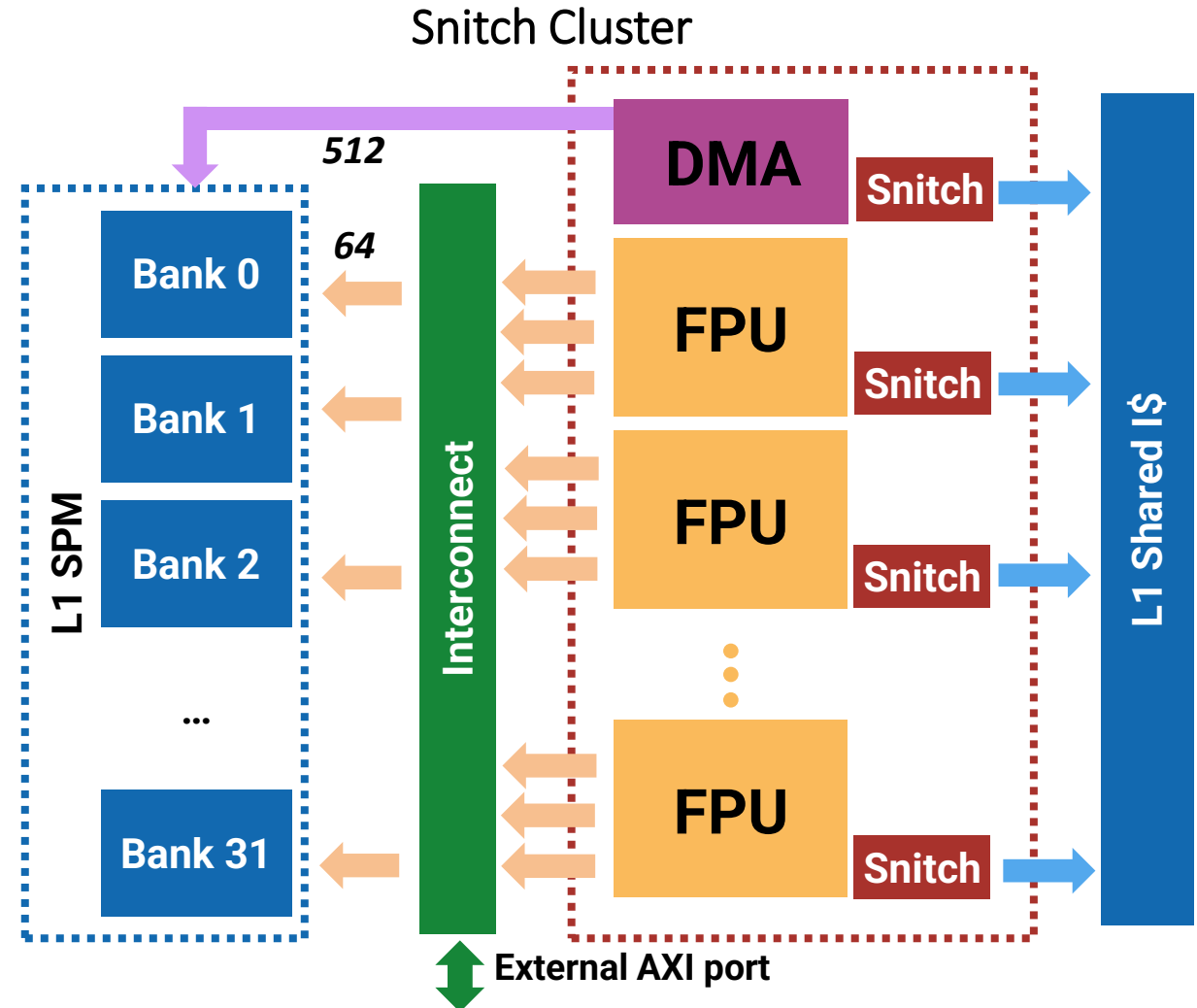




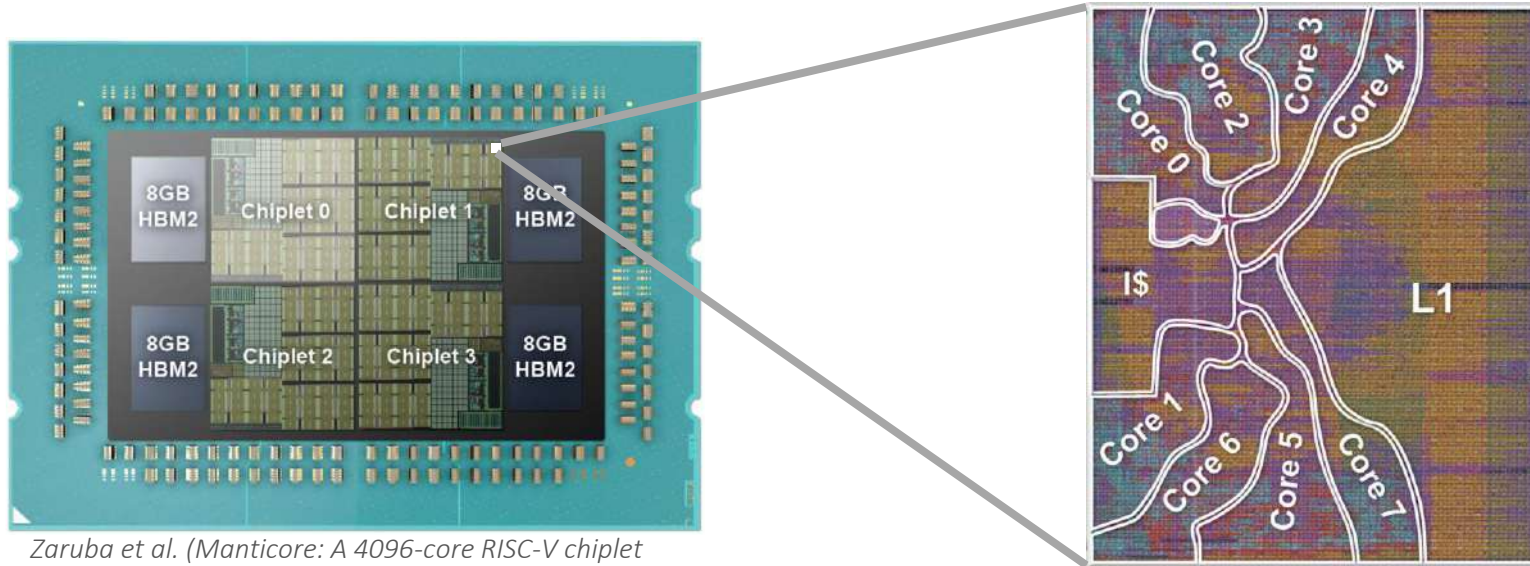
# Efficient RISC-V Compute Clusters: Scalar + Parallel



- Manticore cluster: **Snitch** compute cluster\*
- **Small** integer **scalar** 32b **cores** coupled with **large SIMD FPUs** (FP64, FP32) sharing a scratchpad memory
- **ISA extensions** that implicitly encode loads/stores to register reads/writes + loops handled in HW → ~90% FPU utilization
- Need for narrow FP formats and expanding instructions to efficiently tackle mixed, low-precision NN training



# Academic Architecture for HPC - Manticore



Zaruba et al. (Manticore: A 4096-core RISC-V chiplet architecture for ultraefficient floating-point computing)

- Manticore: a chiplet-based hierarchically-scalable architecture
- Linux-capable host + large manycore accelerator built upon the replication of efficient L1-shared compute clusters

# Without Mixed-Precision Capabilities



In the case of **no mixed-precision** support:

If the application can work on FP16/FP16ALT inputs but accumulation in FP32 required:

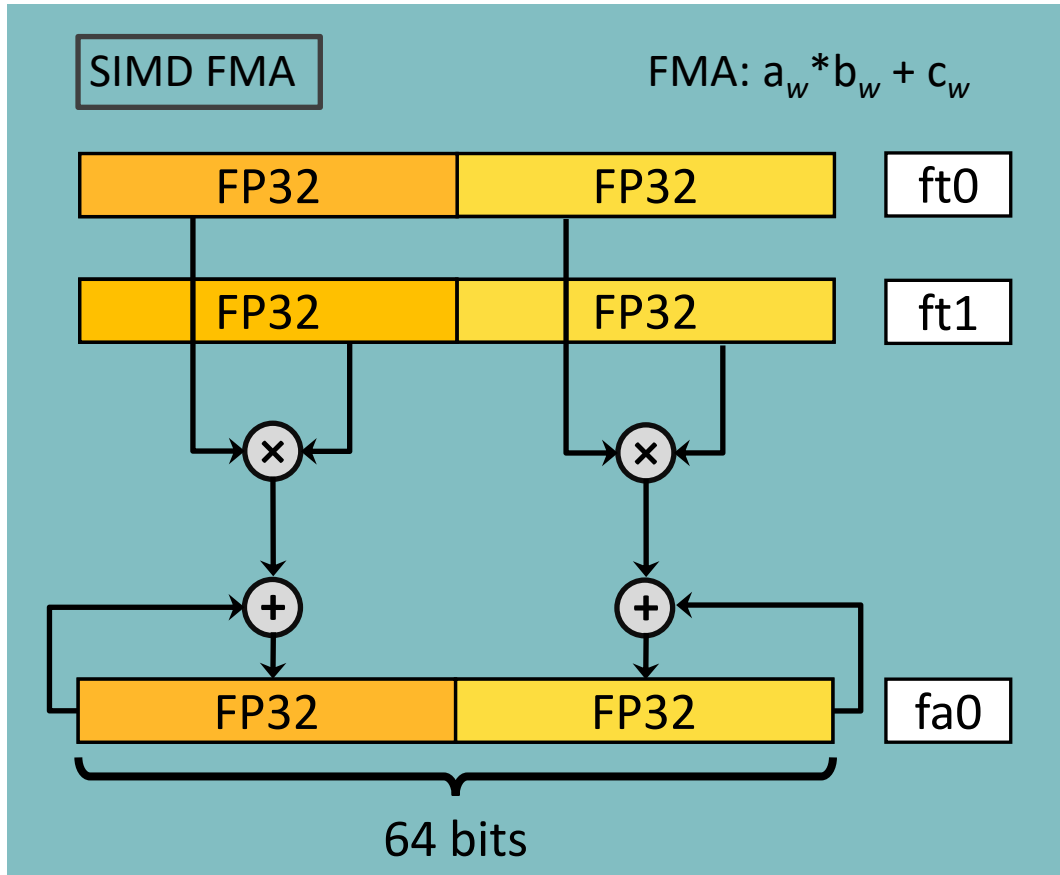
- FP16 with accumulation on FP32 requires additional conversion which affects the performance
- An FP32 kernel can be used

```
<configure loop>
Loop:
    simd_fmacc.s fa0, ft0, ft1 } N
EndLoop:
    <reduction>
```

# Without Mixed-Precision Capabilities



- SIMD FP32 FMA



In the case of **no mixed-precision** support:

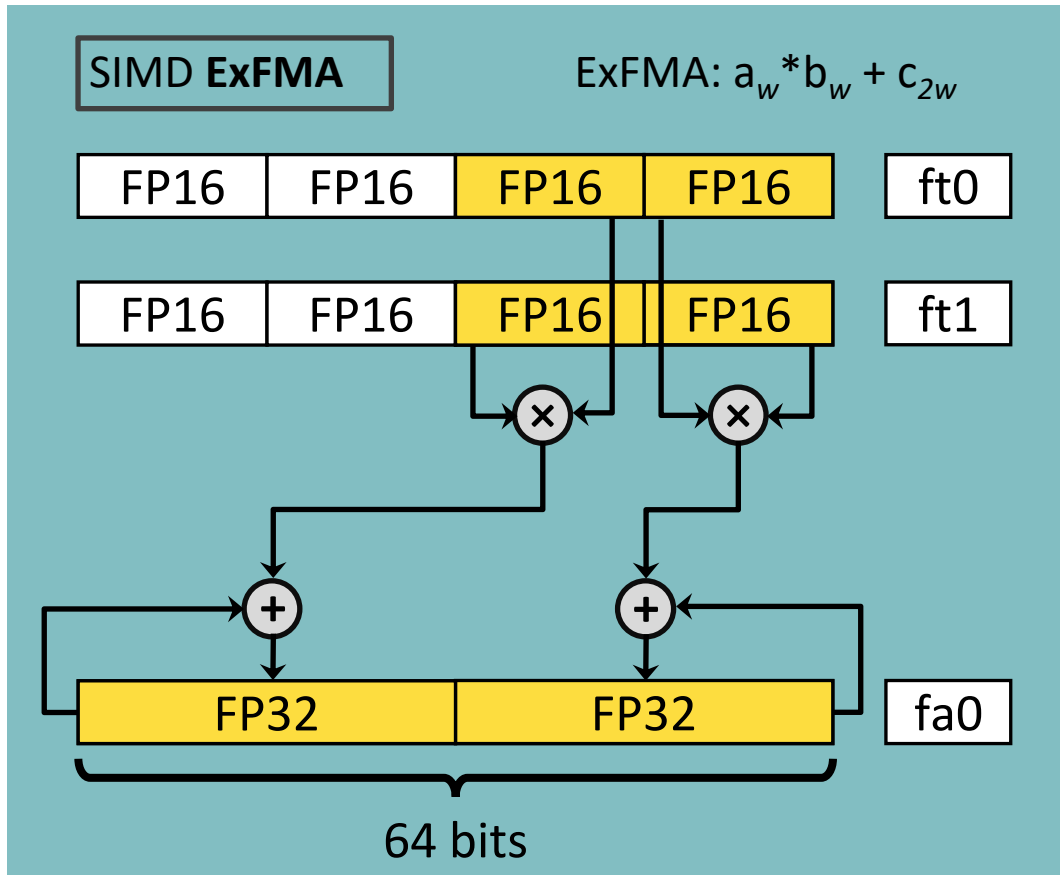
If the application can work on FP16/FP16ALT inputs but accumulation in FP32 required:

- FP16 with accumulation on FP32 requires additional conversion which affects the performance
- An FP32 kernel can be used

```
<configure loop>
Loop:
    simd_fmacc.s fa0, ft0, ft1 } N
EndLoop:
    <reduction>
```

# SIMD Expanding FMA

- SIMD Expanding FMA: **Unbalanced**
- Consumes half of ft0, ft1 and the whole fa0

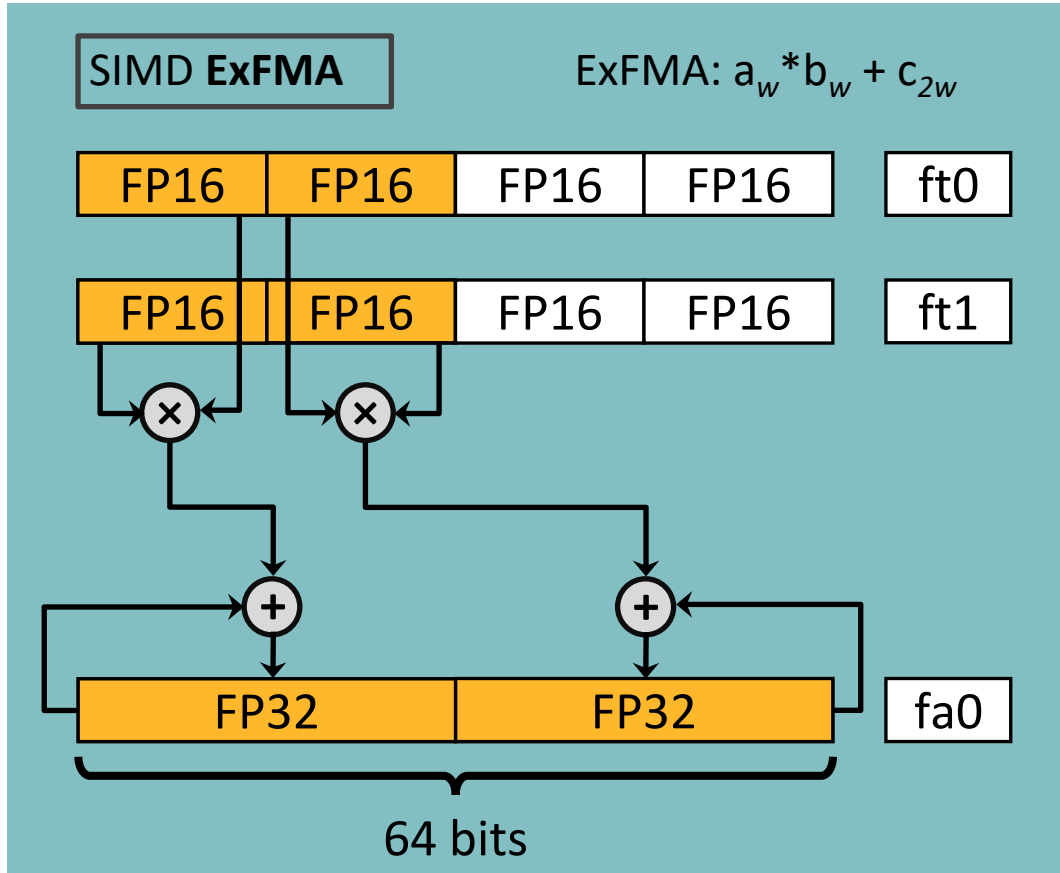


```
<configure loop>  
Loop:  
    simd_exfma.a fa0, ft0, ft1  
    simd_exfma.b fa1, ft0, ft1  
EndLoop:  
    <reduction>
```

N/2  
N/2

# SIMD Expanding FMA

- SIMD Expanding FMA: **Unbalanced**
- Consumes half of ft0, ft1 and the whole fa0



```
<configure loop>
```

```
Loop:
```

```
  simd_exfma.a fa0, ft0, ft1
```

N/2

```
  simd_exfma.b fa1, ft0, ft1
```

N/2

```
EndLoop:
```

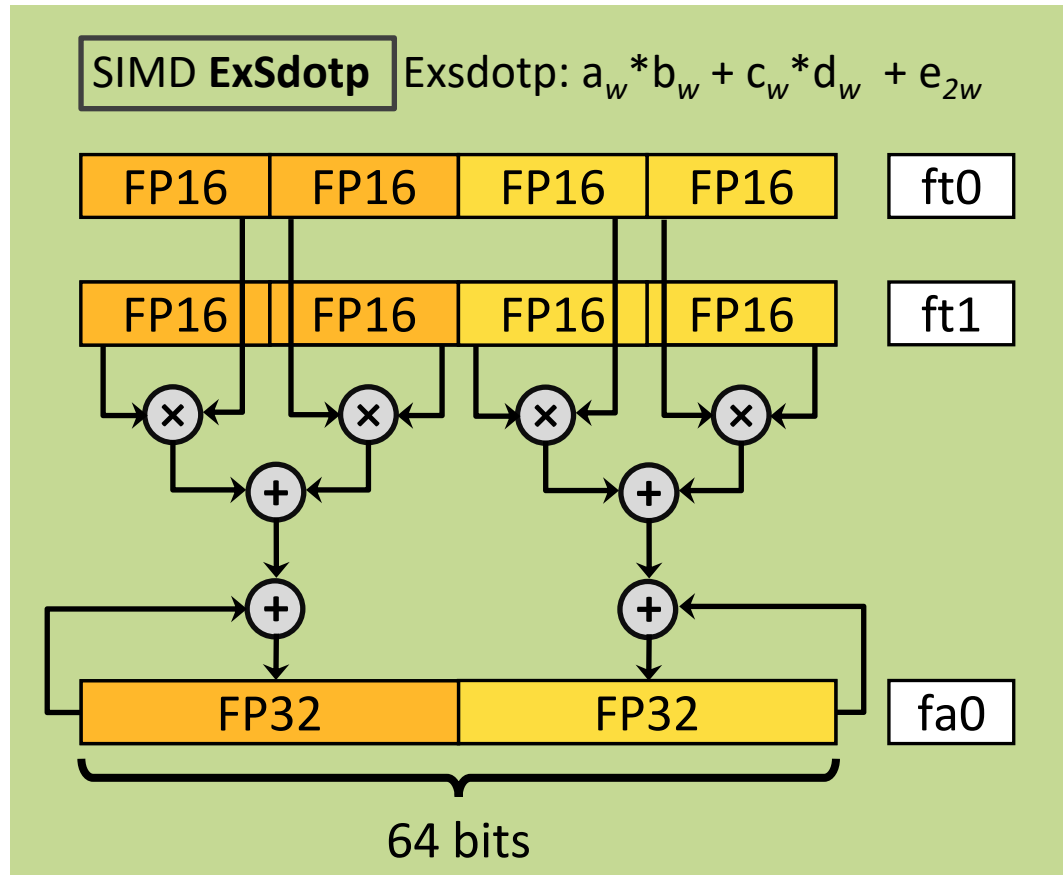
```
  <reduction>
```

- Multiple instructions to cover all possible source locations needed
- The **unbalanced** ExFMA **underutilizes the FPU bandwidth**  
(2\*32+64 bits instead of 3\*64 bits)
- We can provide the FPU with more data and compute more every cycle

# SIMD Expanding Sum of Dot Product for Mixed Precision



- With SIMD ExSdotp, 2x FLOP per cycle
- Same perf as SIMD FP16 FMA but more accurate

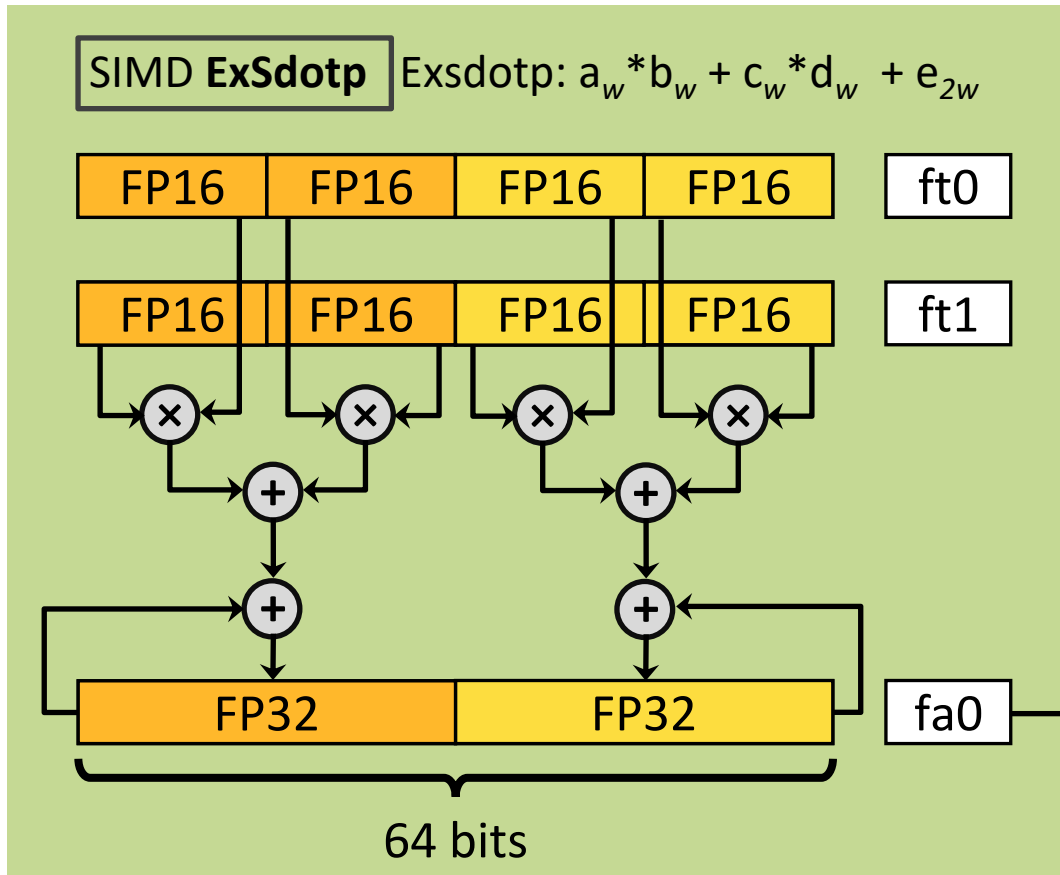


```
<configure loop>  
Loop:  
    exsdotp.s.h fa0, ft0, ft1 } N/2  
EndLoop:  
    vsum.s    fa1, fa0 #reduction
```

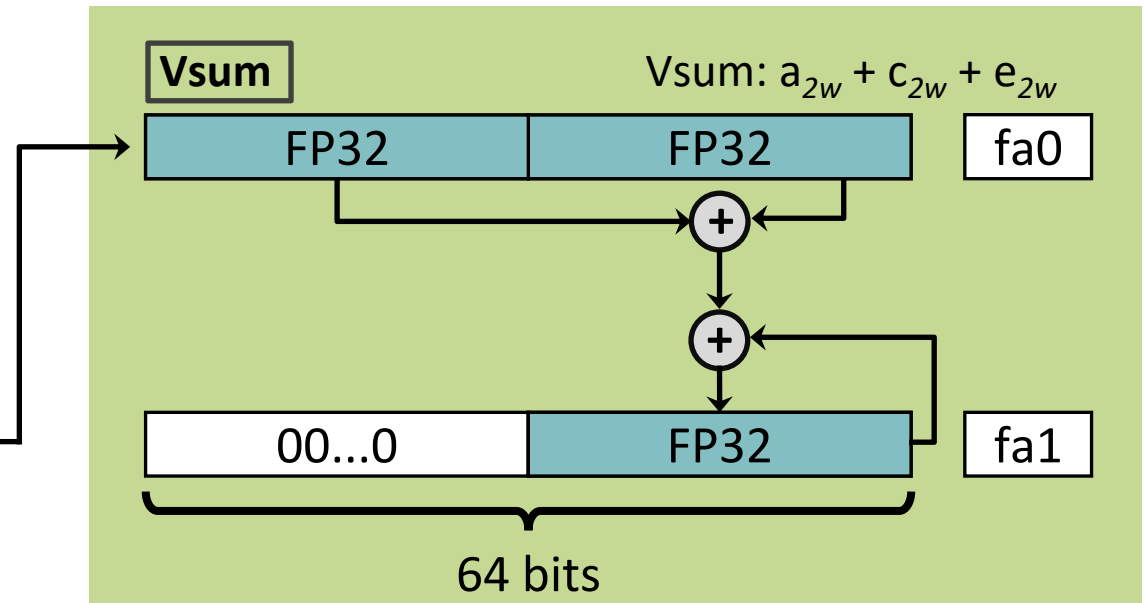
# SIMD Expanding Sum of Dot Product for Mixed Precision



- With SIMD ExSdotp, 2x FLOP per cycle
- Same perf as SIMD FP16 FMA but more accurate

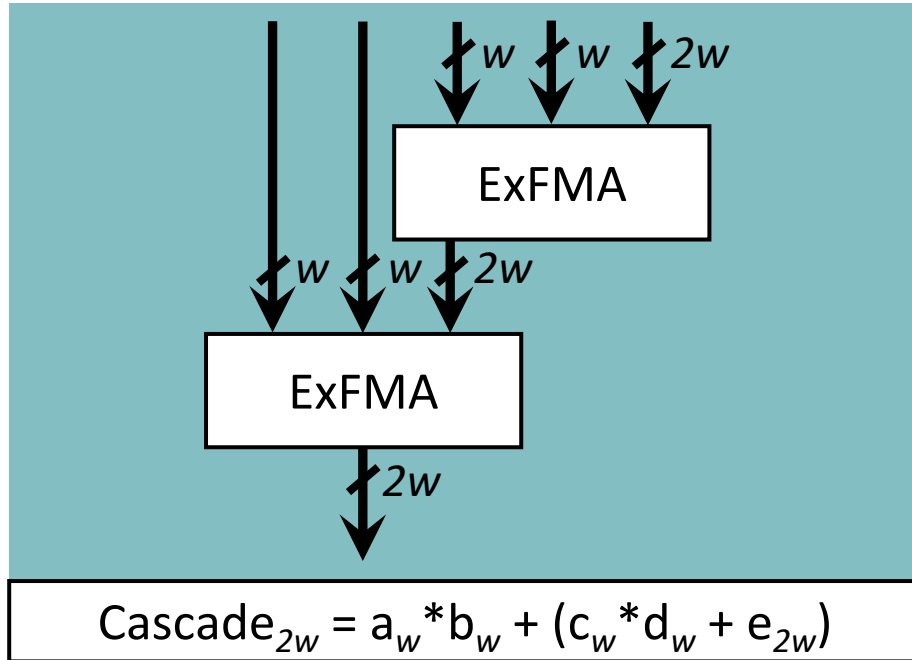


```
<configure loop>  
Loop:  
    exsdotp.s.h fa0, ft0, ft1 } N/2  
EndLoop:  
    vsum.s fa1, fa0 #reduction
```

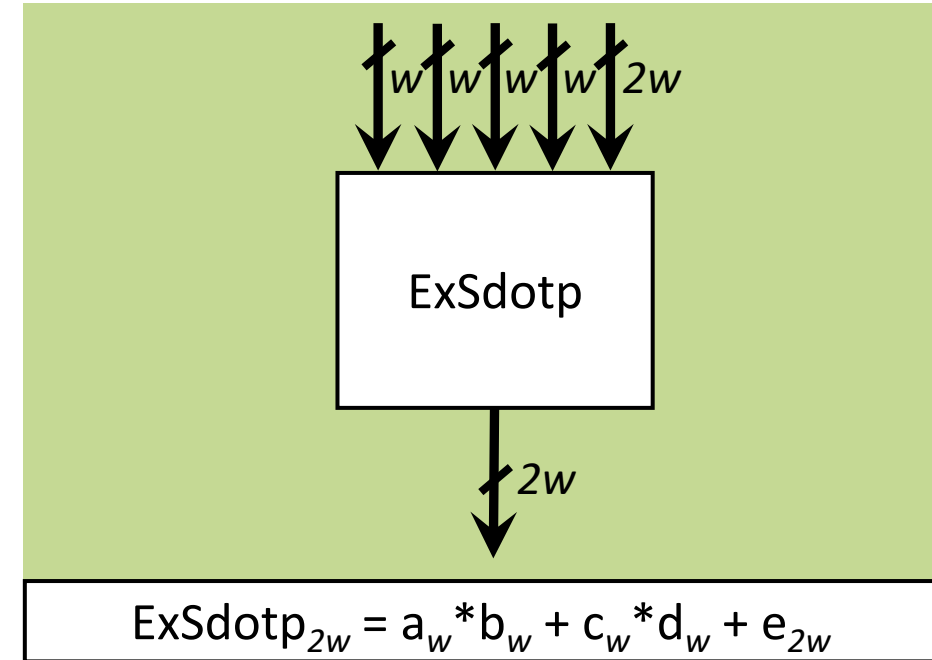




# Why Fused ExSdotp Units?

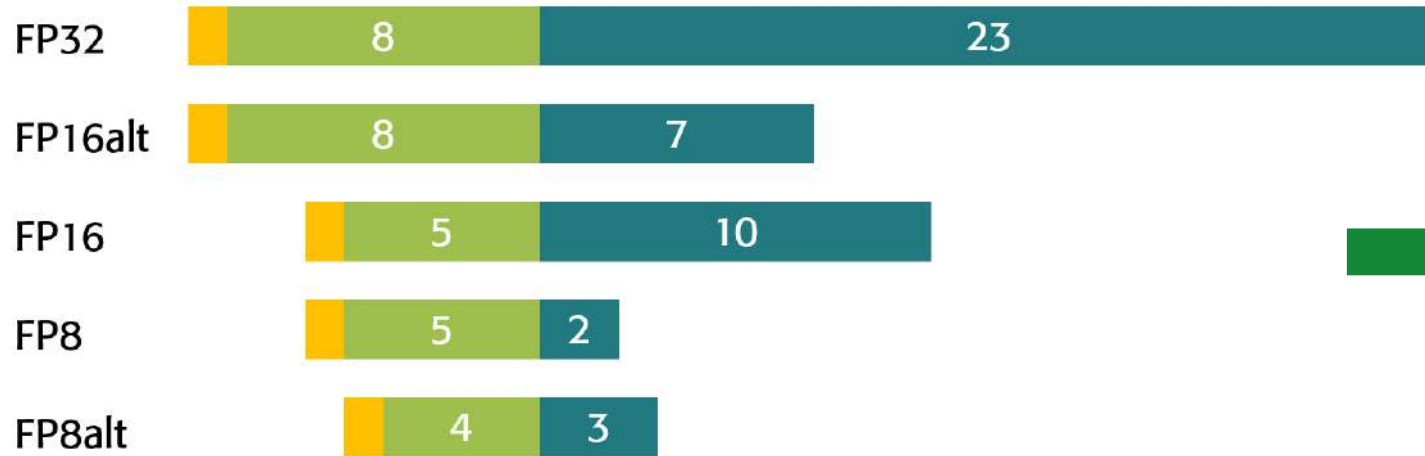


- A cascade of two ExFMAs computes an expanding dot product
- Non-distributive FP addition



- Fused ExSdotp unit
- Single normalization and rounding step
- Opportunity to mitigate issues related to the non-associativity of the two consecutive additions

# Targeted Floating-Point Formats



Many formats + mixed-precision would result in a large ISA extension

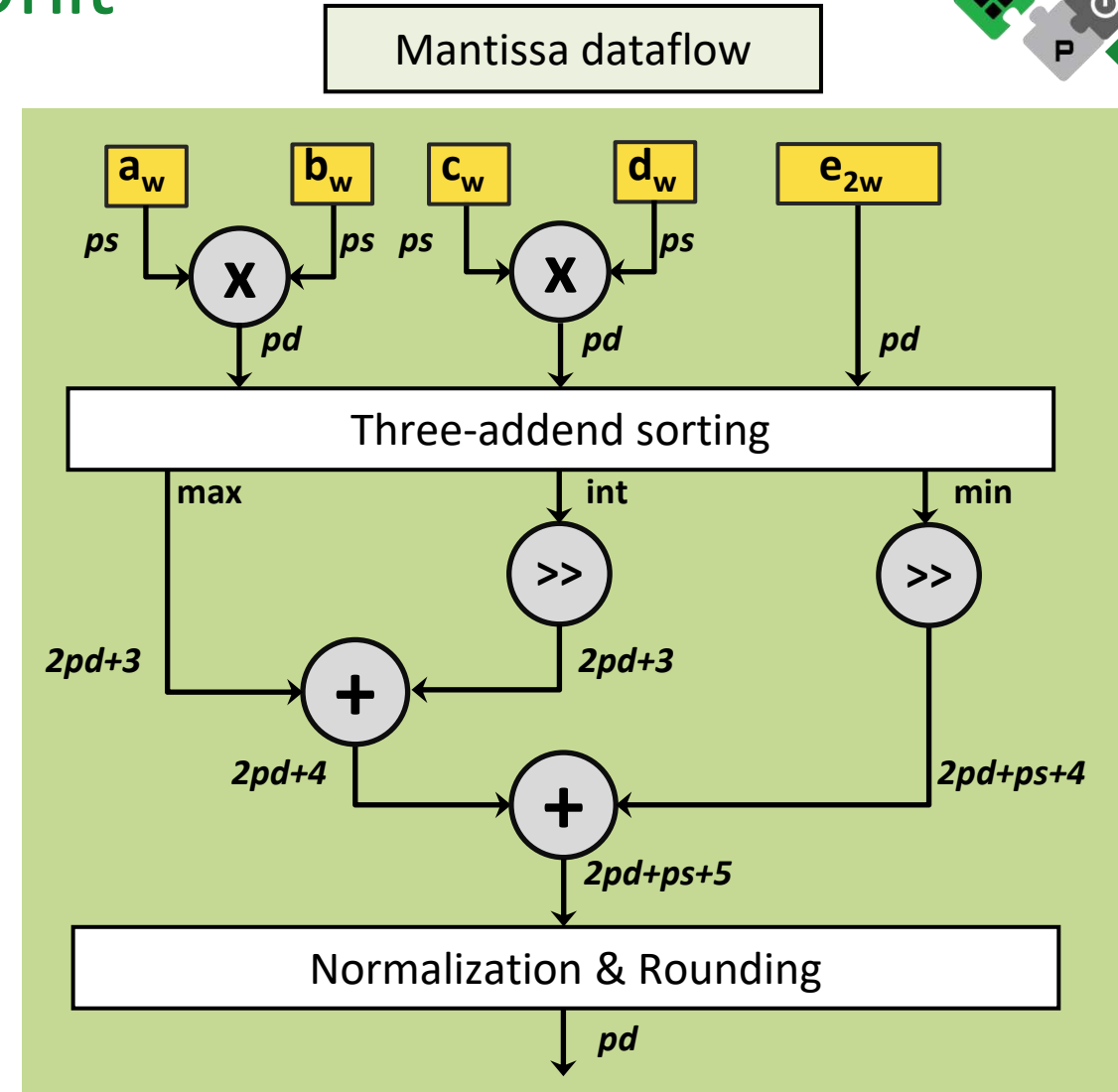
Alternate formats are enabled by FCSRs to reduce the number of instructions

- A parametric design to enable fast exploration of new FP formats
- ExSdotp source formats:
  - FP16alt (1, 8, 7)
  - FP16 (1, 5, 10)
  - FP8alt (1, 5, 2)
  - FP8 (1, 4, 3)
- ExSdotp destination formats:
  - FP32 (1, 8, 23)
  - FP16alt (1, 8, 7)
  - FP16 (1, 5, 10)

# Expanding Sum of Dot Product Unit



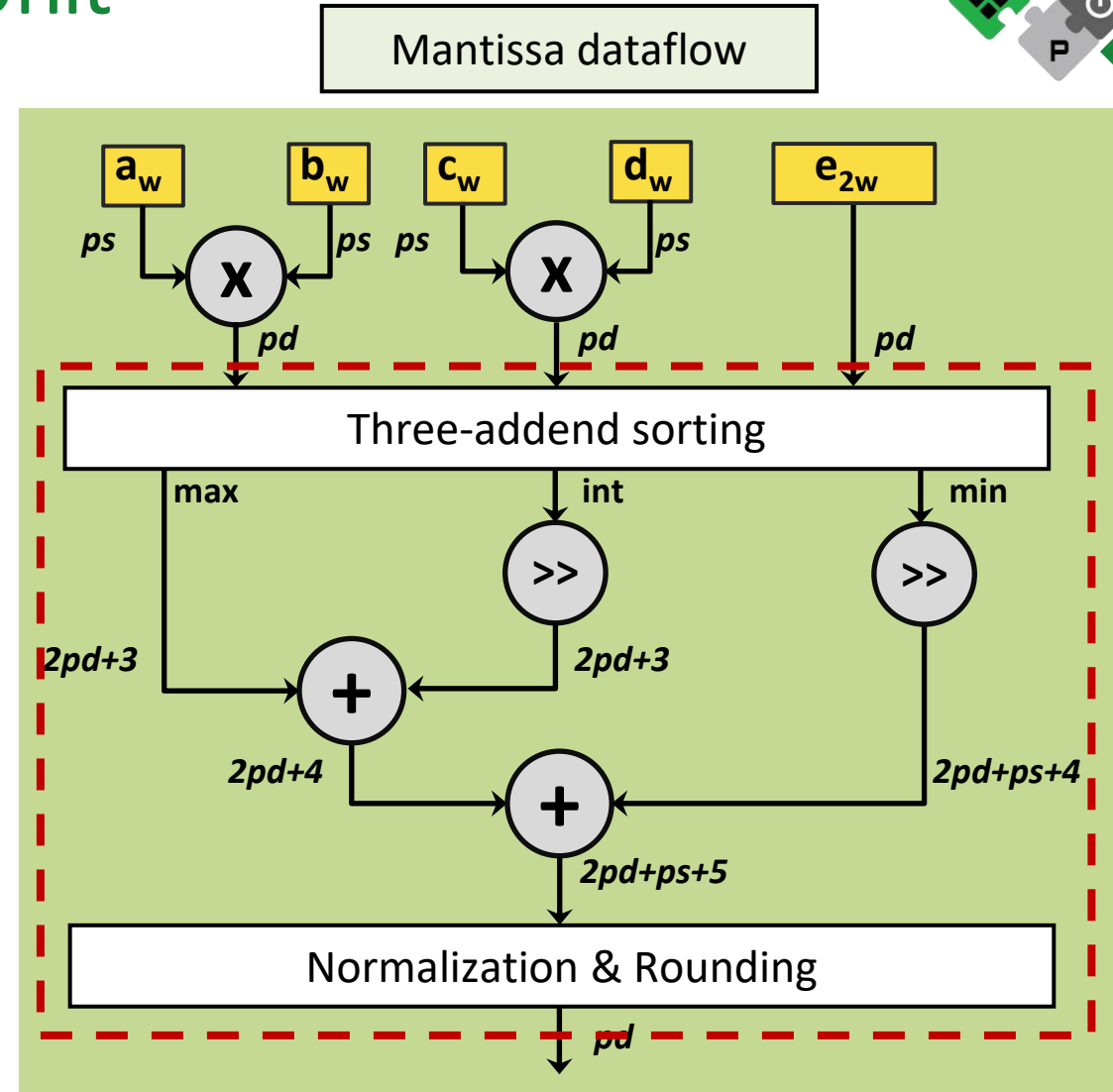
- $w$  = source format bitwidth;  $ps$  = source format mant bits
- $2w$  = destination format bitwidth;  $pd$  = destination format mant bits
- $\text{ExSdotp} = a_w * b_w + c_w * d_w + e_{2w}$



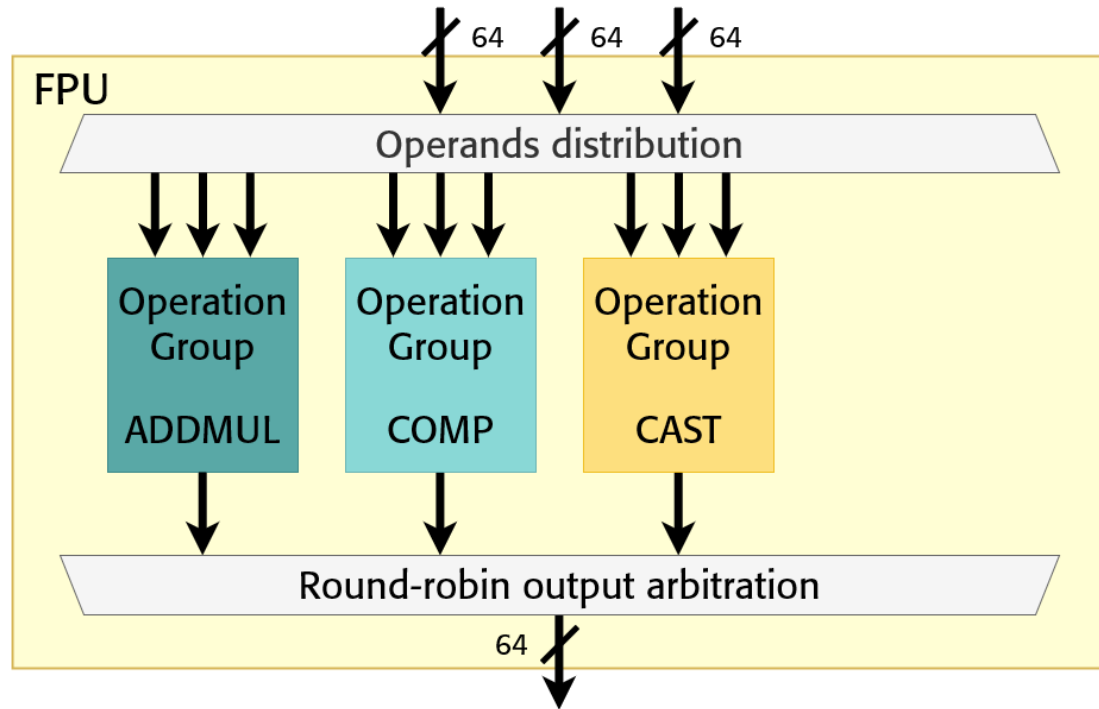
# Expanding Sum of Dot Product Unit



- $w$  = source format bitwidth
- $2w$  = destination format bitwidth
- $\text{ExSdotp} = a_w * b_w + c_w * d_w + e_{2w}$
- $\text{ExVsum} = a_w * 1 + c_w * 1 + e_{2w}$
- $\text{Vsum} = a_{2w} + c_{2w} + e_{2w}$
- ExVsum/Vsum to reduce and accumulate the results packed in a register after SIMD ExSdotp executions
- Support for non-expanding three-term sum added by **bypassing** the multiplications
- All the necessary logic is already present as the targeted ExSdotp operations were expanding

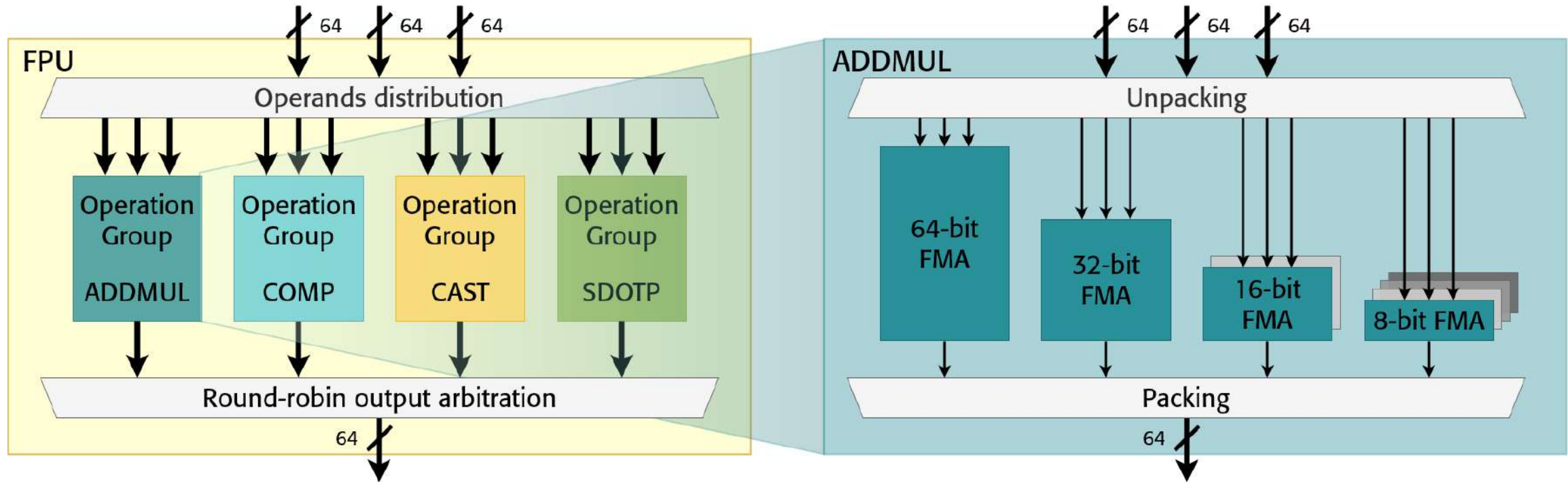


# Enhancing CVFPU with SIMD ExSdotp



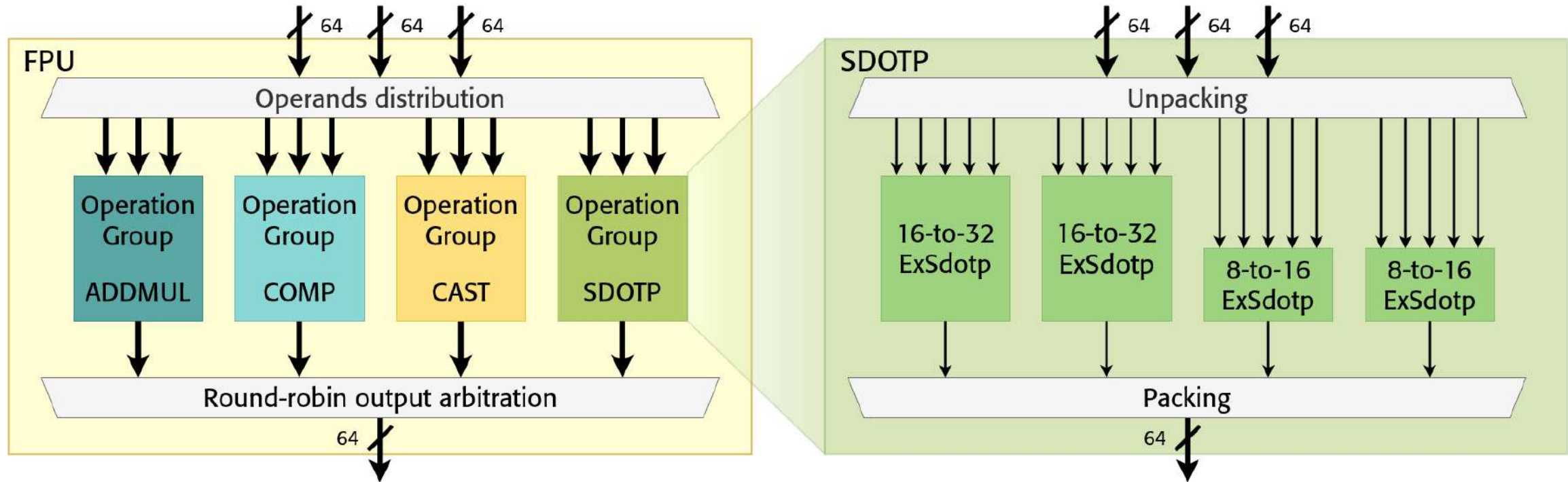
- CVFPU is a highly-parameterized open-source **modular** energy-efficient multi-format FPU

# Enhancing CVFPU with SIMD ExSdotp



- CVFPU is a highly-parameterized open-source **modular** energy-efficient multi-format FPU
- SIMD FMA unit
- As proposed in <https://iis-git.ee.ethz.ch/smach/smallFloat-spec>

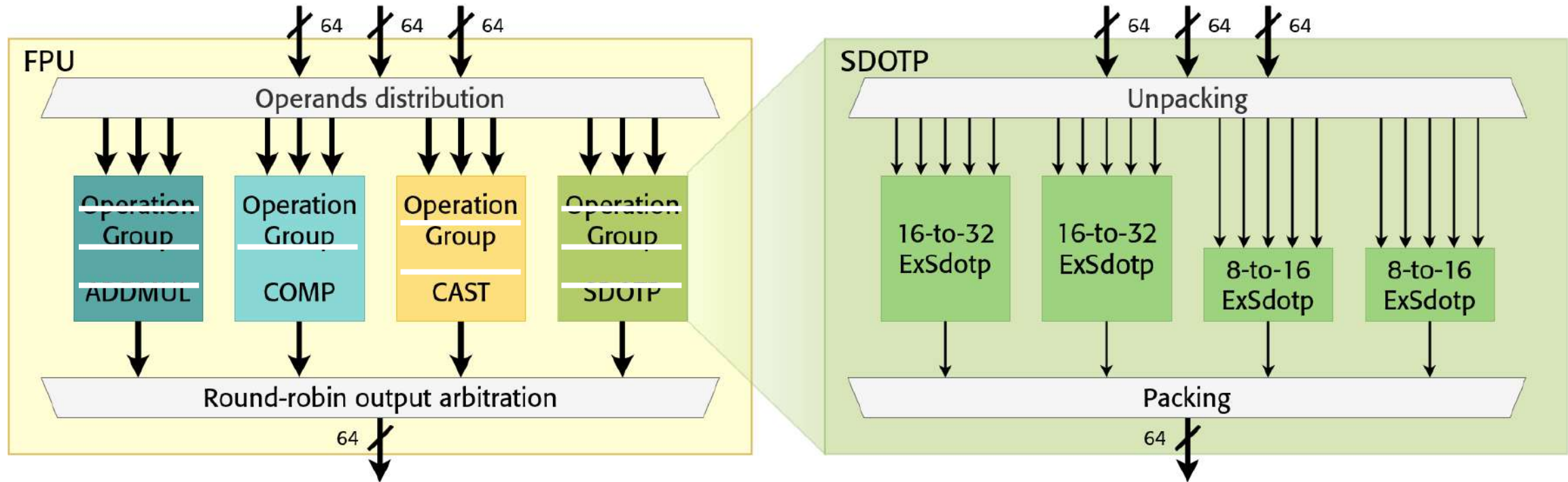
# Enhancing CVFPU with SIMD ExSdotp



- CVFPU is a highly-parameterized open-source **modular** energy-efficient multi-format FPU
- SIMD ExSdotp unit integrated into CVFPU as a new **operation group** block
- SIMD SDOTP: **two** 16-to-32-bit units and **two** 8-to-16-bit units
- Up to **two** 16-to-32-bit ExSdotp and **four** 8-to-16-bit ExSdotp per cycle



# Enhancing CVFPU with SIMD ExSdotp

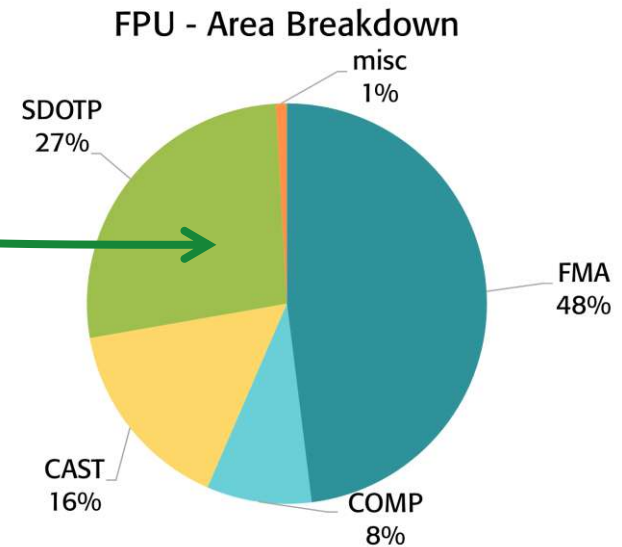
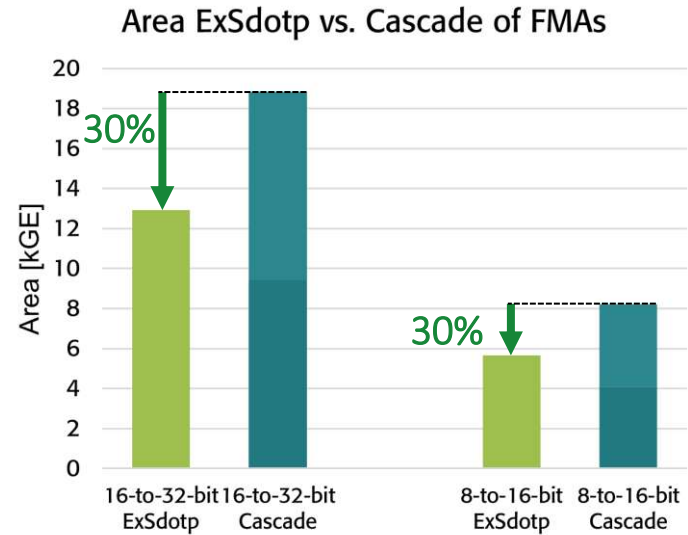


- **Parametrizable number of pipeline levels.** In our specific case, we selected:
  - SDOTP: 3 levels of pipeline registers
  - ADDMUL: 3 levels of pipeline registers
  - CAST: 2 levels of pipeline registers
  - COMP: 1 levels of pipeline registers

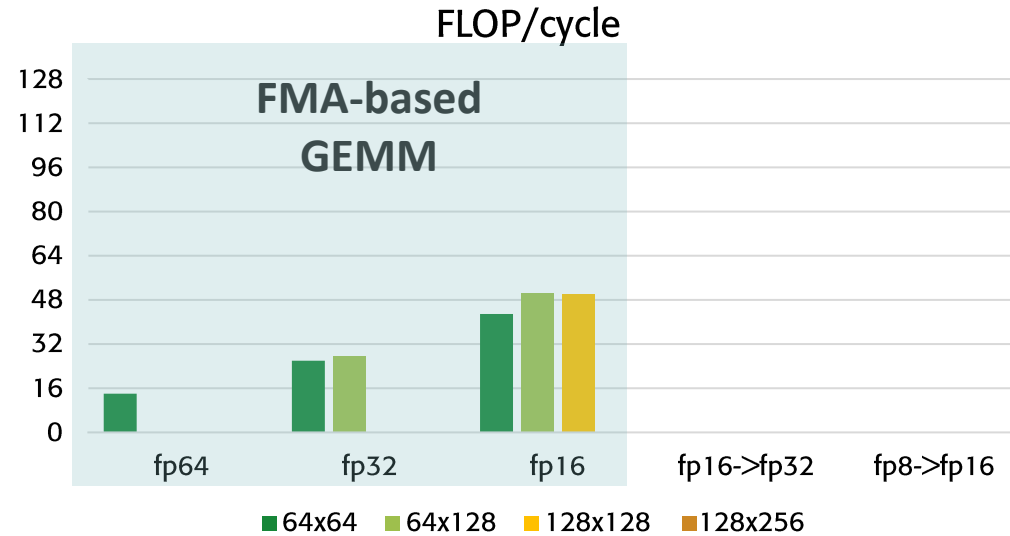


# ExSdotp & CVFPU: Area and Timing

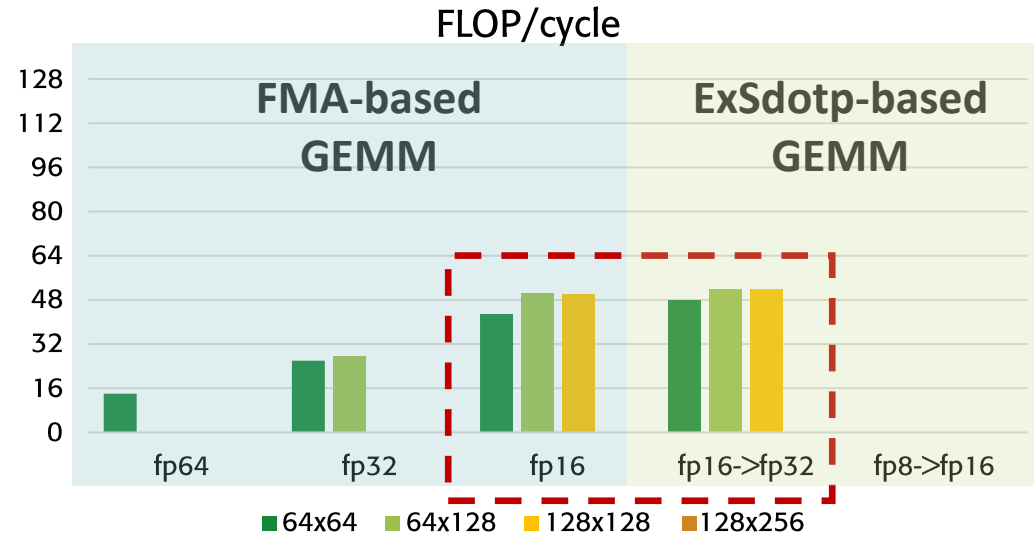
- Implemented in GlobalFoundries 12nm FinFET technology
- Max Frequency → 1.24GHz (typ 0.8V, 25 °C)
- The fused ExSdotp unit allows for around **30% area** and **critical path reduction** with respect to a cascade of ExFMA modules.
- The SIMD SDOTP unit occupies **44.5 kGE**, amounting to **27%** of the enhanced FPU area (overall FPU area = 165kGE).
- Full extension introduced **less than 15% area overhead at a cluster level**



# MiniFloat-NN Cluster: Performance and Efficiency

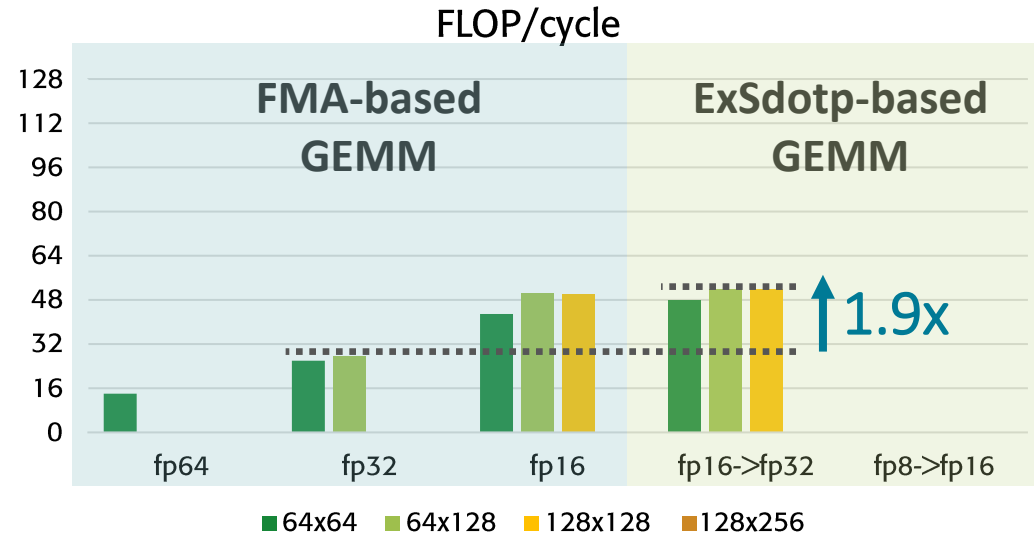


# MiniFloat-NN Cluster: Performance and Efficiency



Same performance at a higher accumulation precision

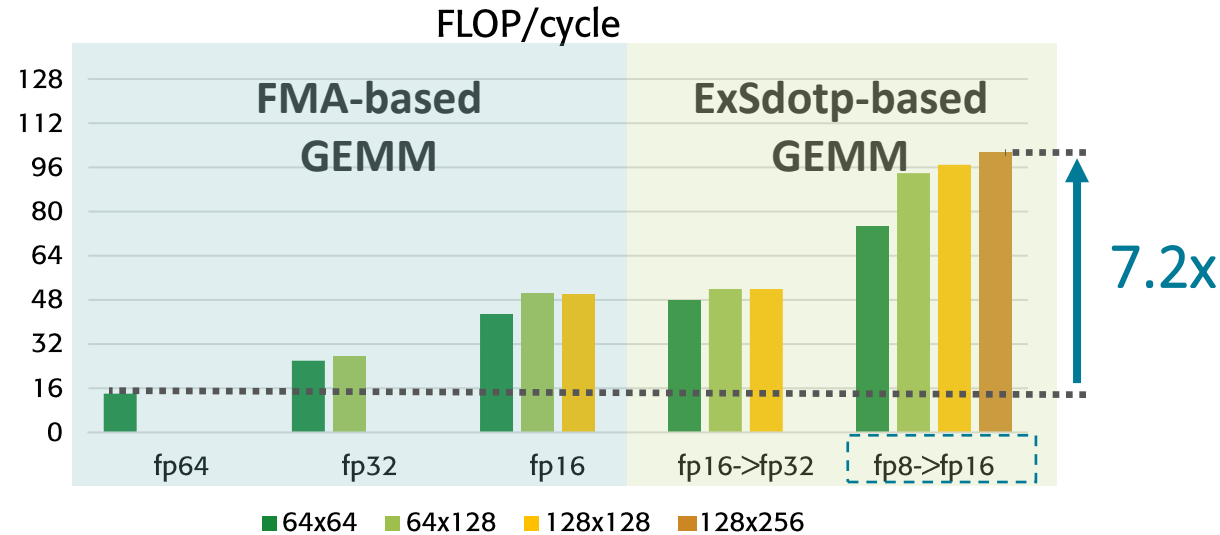
# MiniFloat-NN Cluster: Performance and Efficiency



# MiniFloat-NN Cluster: Performance and Efficiency



More than 1.9x  
performance  
wrt ExFMA and  
50% more  
efficient



With the  
required higher  
precision  
accumulation

More than **7x performance** and **energy efficiency** improvement with respect to FP64 computation (<15% area overhead on the entire cluster) + **reduced memory footprint**

# Conclusion



- Transprecision computing enabling high efficiency and performance gains
- Transprecision support allows for exploiting lower memory footprint
- Open-source, highly efficient and flexible FPU enabling transprecision computing on general-purpose architecture

<https://github.com/pulp-platform/fpnew>

# Conclusion



10+ ASICs including FPnew







# Not only general-purpose computing...



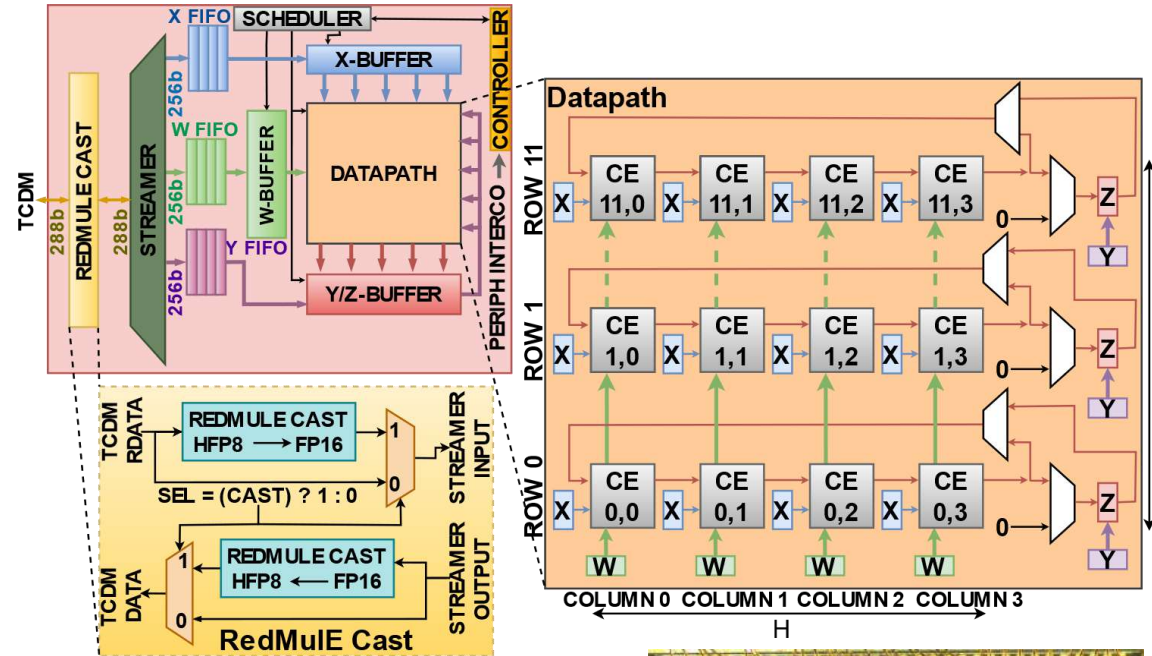
# CVFPU Functional Units as Building Blocks for DSA Datapath



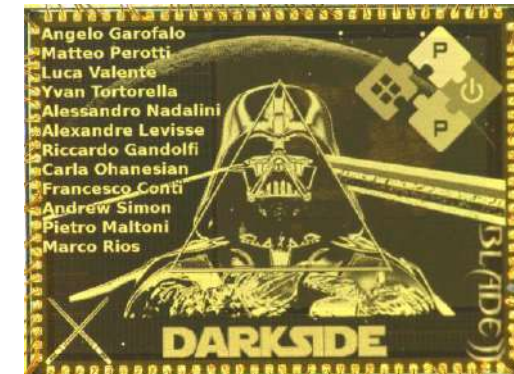
- CVFPU is a modular design
- The execution units can be reused as building blocks for domain-specific accelerator (DSA) datapaths.
- Example → **RedMule**: floating-point GEMM accelerator (<https://github.com/pulp-platform/redmule>)
- RedMule**: HW Accelerator for GEMMs for FP8/FP16
  - 2D Array of Computing Elements (CE) operating in **lockstep** and distributed in rows and columns
  - CEs has private copies of their **X**-matrix elements, **W**-matrix elements broadcasted among rows of CEs.
  - In/output cast unit casting (FP8 ↔ FP16) for high computing accuracy

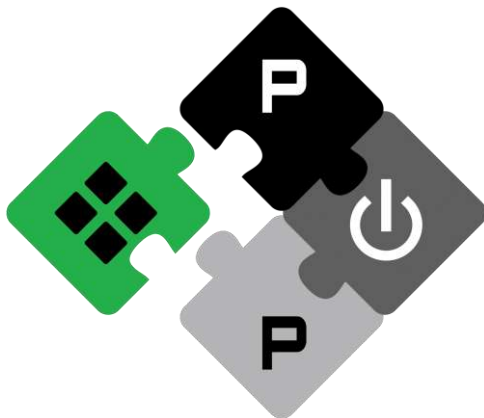


$$GEMM: Z = (X \times W) + Y$$




**Darkside 65nm, ESSCIRC'22**  
(UNIBO+ETHZ+EPFL)





Thank you for your attention!

@pulp\_platform 

pulp-platform.org 

youtube.com/pulp\_platform 