Floating-Point Arithmetic Paradigms for High-Performance Computing: Software Algorithms and Hardware Designs

LEDOUX Louis

Universitat Politècnica de Catalunya (UPC)
Barcelona Supercomputing Center (BSC)

Advisors:
Dr. Marc Casas
Dr. Eduard Ayguadé

Thesis Defense
October 2, 2024

Agenda

1. Introduction
2. Background
3. Accelerating Deep Learning Inference with the Posit Number System
4. A Generator of Numerically-Tailored Accelerators for Batched GEMMs
5. An Open-Source Framework for Efficient Numerically-Tailored Computations
6. Divisions in Vector Processing: Leveraging Opportunities from a Paradigm Shift
7. Closing

Challenges Addressed in This Thesis

A. The Walls

Challenges in the "Dark Silicon" era, the slowdown of Moore's Law, the stalling of Dennard Scaling, the Power Wall, and the Memory Wall.

B. Numerical Sparsity of Workloads

The necessity ranges from **2** bits in Large Language Models to **1000** bits in experimental mathematics.

C. "Communication Dominates Arithmetic"

The majority of time is spent **moving data**, which **costs orders of magnitude more than computation**.

A. The Walls

Dark Silicon and Power Wall

Dark Silicon, restricting the simultaneous activation of transistors and pushing towards **specialized**, rather than **general-purpose**, computing solutions [SGH+14].

Memory Wall

The growing gap between processor speeds and memory latency severely limits system performance [DGY+99].

Arithmetic Perspective

General Purpose Floating-Point / FPU goes against **specialization**. Floating-Point **Performance** impacted by the **Memory Wall**.

A. The Walls

Dark Silicon and Power Wall

Dark Silicon, restricting the simultaneous activation of transistors and pushing towards **specialized**, rather than **general-purpose**, computing solutions [SGH+14].

Memory Wall

The growing gap between processor speeds and memory latency severely limits system performance [DGY+99].

Arithmetic Perspective

General Purpose Floating-Point / FPU goes against **specialization**.
Floating-Point **Performance** impacted by the **Memory Wall**.

A. The Walls

Dark Silicon and Power Wall

Dark Silicon, restricting the simultaneous activation of transistors and pushing towards **specialized**, rather than **general-purpose**, computing solutions [SGH+14].

Memory Wall

The growing gap between processor speeds and memory latency severely limits system performance [DGY+99].

Arithmetic Perspective

General Purpose Floating-Point / FPU goes against **specialization**. Floating-Point **Performance** impacted by the **Memory Wall**.

B. Numerical Sparsity of Workloads

Precision-Hungry Workloads

E.g. scientific computing, quantum calculations, and detailed simulations, where high precision ensures **accuracy and reproducibility** [BM13, ZYD96, EGK+09, LQR96].

Precision-Resilient Workloads

Emerging deep learning models demonstrate robustness to reduced precision, optimizing computational resources [Joh18, CHS+16].

General-Purpose Computing Challenge

Developers are often limited to a few floating-point formats, compilation flags, and Floating-Point Units (FPU).

B. Numerical Sparsity of Workloads

Precision-Hungry Workloads

E.g. scientific computing, quantum calculations, and detailed simulations, where high precision ensures **accuracy and reproducibility** [BM13, ZYD96, EGK+09, LQR96].

Precision-Resilient Workloads

Emerging deep learning models demonstrate robustness to reduced precision, optimizing computational resources [Joh18, CHS+16].

General-Purpose Computing Challenge

Developers are often limited to a few floating-point formats, compilation flags, and Floating-Point Units (FPU).

B. Numerical Sparsity of Workloads

Precision-Hungry Workloads

E.g. scientific computing, quantum calculations, and detailed simulations, where high precision ensures **accuracy and reproducibility** [BM13, ZYD96, EGK+09, LQR96].

Precision-Resilient Workloads

Emerging deep learning models demonstrate robustness to reduced precision, optimizing computational resources [Joh18, CHS+16].

General-Purpose Computing Challenge

Developers are often limited to a few floating-point formats, compilation flags, and Floating-Point Units (FPU).

C. "Communication dominates Arithmetic"

60% of time spent in communication [BRM+18].

Integer Operation	# of Operations	Latency (Clock Cycle)
Add	12	1
Multiply	4	1
Memory	**Size (KBytes)**	**Latency (Clock Cycle)**
L1 Data Cache	32	4 - 5
L2 Cache	256	12
L3 Cache	8192	36 - 58
DRAM	-	230 - 422

Latency comparison [Hor14].

Arithmetic Operation	Bit Width	Energy (pJ)	Normalized Energy (per bit)
Integer Add	32	0.1	1
Integer Multiply	32	3.1	31
Float Add (16-bit)	16	0.4	8
Float Add (32-bit)	32	0.9	9
Float Multiply (16-bit)	16	1.1	22
Float Multiply (32-bit)	32	3.7	37
Memory	**Bit Width**	**Energy (pJ)**	**Normalized Energy (per bit)**
L1 Data Cache	64	20	100
DRAM	64	1300 - 2600	6500 - 13000

Energy consumption [Hor14].

"Arithmetic wall"

Definition of the Arithmetic wall

Collectively, these problems exhibit the **gap** between the **arithmetic solutions** and the **HPC challenges**

How This Thesis Addresses Key Challenges

I. Format Adaptability

Beyond IEEE754: adaptative formats for more tasks. **Maximize the entropy** per datum and avoid **leaving the chips for memory**.
targets: A, B, and C.

II. Arithmetic Efficiency

Every bit and wire count in the ALU. Toward **necessary** and **sufficient** circuitry.
targets: mainly A, but C.

III. Bridging Software and Hardware

Providing cohesive solutions that span from high-performance computing (HPC) libraries to silicon layout.
targets: A, B, and C.

How This Thesis Addresses Key Challenges

I. Format Adaptability

Beyond IEEE754: adaptative formats for more tasks. **Maximize the entropy** per datum and avoid **leaving the chips for memory**.
targets: A, B, and C.

II. Arithmetic Efficiency

Every bit and wire count in the ALU. Toward **necessary** and **sufficient** circuitry.
targets: mainly A, but C.

III. Bridging Software and Hardware

Providing cohesive solutions that span from high-performance computing (HPC) libraries to silicon layout.
targets: A, B, and C.

How This Thesis Addresses Key Challenges

I. Format Adaptability

Beyond IEEE754: adaptative formats for more tasks. **Maximize the entropy** per datum and avoid **leaving the chips for memory**.
targets: A, B, and C.

II. Arithmetic Efficiency

Every bit and wire count in the ALU. Toward **necessary** and **sufficient** circuitry.
targets: mainly A, but C.

III. Bridging Software and Hardware

Providing cohesive solutions that span from high-performance computing (HPC) libraries to silicon layout.
targets: A, B, and C.

Concrete Contributions

First Contribution: Posit AI acceleration

"Accelerating Deep Learning Inference with the Posit Number System" - *OpenPOWER Summit 2019*, focuses on **I**, **II**.

Second Contribution: Generator of Systolic Arrays

"A Generator of Numerically-Tailored Accelerators for Batched GEMMs" - *FCCM2022*, focuses on **I**, **II**.

Third Contribution: Numerically-tailored BLAS

"An Open-Source Framework for Efficient Numerically-Tailored Computations" - *FPL2023*, focuses on **III**.

Last Contribution: Floating-Point Divisions

"Divisions in Vector Processing: Leveraging Opportunities from a Paradigm Shift" - *DATE/OSDA2024 & WIP*, focuses on **II**.

Concrete Contributions

First Contribution: Posit AI acceleration

"Accelerating Deep Learning Inference with the Posit Number System" - *OpenPOWER Summit 2019*, focuses on **I**, **II**.

Second Contribution: Generator of Systolic Arrays

"A Generator of Numerically-Tailored Accelerators for Batched GEMMs" - *FCCM2022*, focuses on **I**, **II**.

Third Contribution: Numerically-tailored BLAS

"An Open-Source Framework for Efficient Numerically-Tailored Computations" - *FPL2023*, focuses on **III**.

Last Contribution: Floating-Point Divisions

"Divisions in Vector Processing: Leveraging Opportunities from a Paradigm Shift" - *DATE/OSDA2024 & WIP*, focuses on **II**.

Concrete Contributions

First Contribution: Posit AI acceleration

"Accelerating Deep Learning Inference with the Posit Number System" - *OpenPOWER Summit 2019*, focuses on **I**, **II**.

Second Contribution: Generator of Systolic Arrays

"A Generator of Numerically-Tailored Accelerators for Batched GEMMs" - *FCCM2022*, focuses on **I**, **II**.

Third Contribution: Numerically-tailored BLAS

"An Open-Source Framework for Efficient Numerically-Tailored Computations" - *FPL2023*, focuses on **III**.

Last Contribution: Floating-Point Divisions

"Divisions in Vector Processing: Leveraging Opportunities from a Paradigm Shift" - *DATE/OSDA2024 & WIP*, focuses on **II**.

Concrete Contributions

First Contribution: Posit AI acceleration

"Accelerating Deep Learning Inference with the Posit Number System" - *OpenPOWER Summit 2019*, focuses on **I**, **II**.

Second Contribution: Generator of Systolic Arrays

"A Generator of Numerically-Tailored Accelerators for Batched GEMMs" - *FCCM2022*, focuses on **I**, **II**.

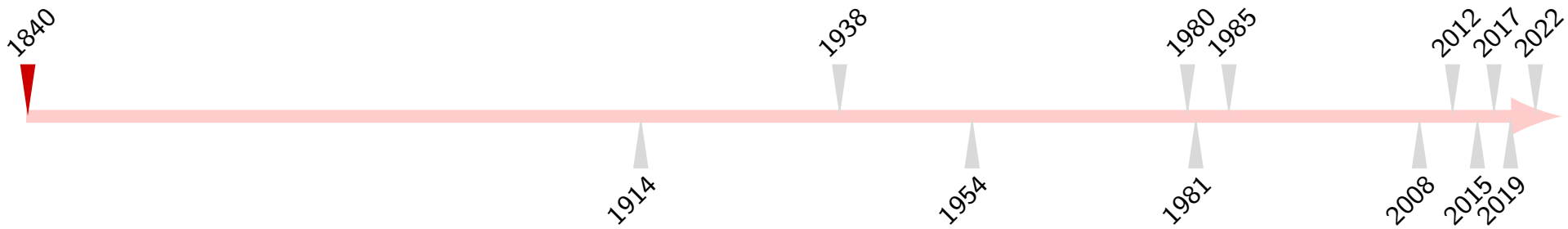
Third Contribution: Numerically-tailored BLAS

"An Open-Source Framework for Efficient Numerically-Tailored Computations" - *FPL2023*, focuses on **III**.

Last Contribution: Floating-Point Divisions

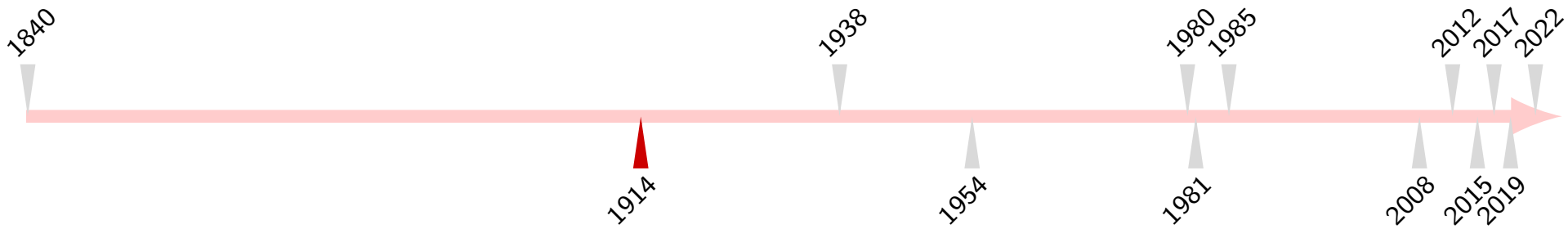
"Divisions in Vector Processing: Leveraging Opportunities from a Paradigm Shift" - *DATE/OSDA2024 & WIP*, focuses on **II**.

Historical Background of Floating-Point Arithmetic



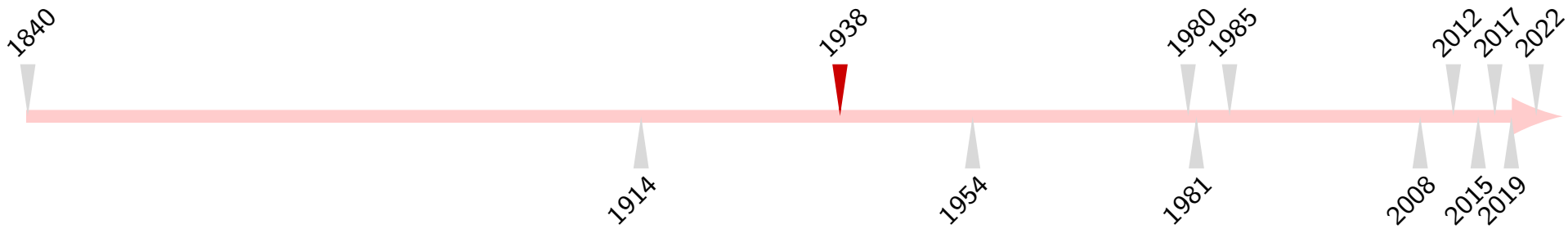
- 1840: Babbage and Lovelace conceptualize the first floating-point.
- 1914: Leonardo T. Quevedo builds an electromechanical calculator with floating-point.
- 1938: Konrad Zuse **Z1**, the first programmable mechanical computer with floating-point arithmetic.
- 1954: IBM 704, the first mass-produced computer with floating-point hardware.
- 1980: Intel introduces the 8087 co-processor.
- 1981: **Kulisch**: "Computer Arithmetic in Theory and Practice"
- 1985: **IEEE754**-85 standard.
- 2008: IEEE754-08 introduces half-precision and FMA
- 2012: AlexNet sparks the AI revolution with its CNN structure.
- 2015: Introduction of bfloat16, optimizing neural network computations.
- 2017: Introduction of the **Posit** format, a new take on floating-point representation.
- 2019: IEEE754 standard evolves to fix errors
- 2022: **LLM revolution**, chatgpt, MX format (e2m2, e4m3, e5m2), tensor cores

Historical Background of Floating-Point Arithmetic



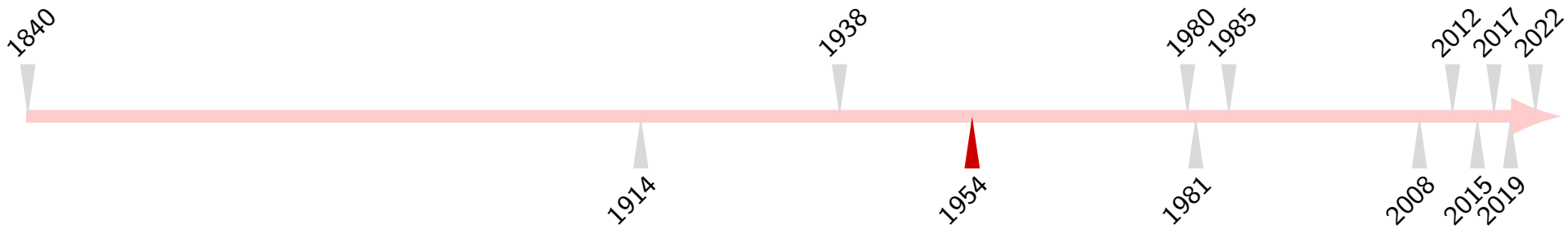
- 1840: Babbage and Lovelace conceptualize the first floating-point.
- 1914: Leonardo T. Quevedo builds an electromechanical calculator with floating-point.
- 1938: Konrad Zuse **Z1**, the first programmable mechanical computer with floating-point arithmetic.
- 1954: IBM 704, the first mass-produced computer with floating-point hardware.
- 1980: Intel introduces the 8087 co-processor.
- 1981: **Kulisch**: "Computer Arithmetic in Theory and Practice"
- 1985: **IEEE754**-85 standard.
- 2008: IEEE754-08 introduces half-precision and FMA
- 2012: AlexNet sparks the AI revolution with its CNN structure.
- 2015: Introduction of bfloat16, optimizing neural network computations.
- 2017: Introduction of the **Posit** format, a new take on floating-point representation.
- 2019: IEEE754 standard evolves to fix errors
- 2022: **LLM revolution**, chatgpt, MX format (e2m2, e4m3, e5m2), tensor cores

Historical Background of Floating-Point Arithmetic



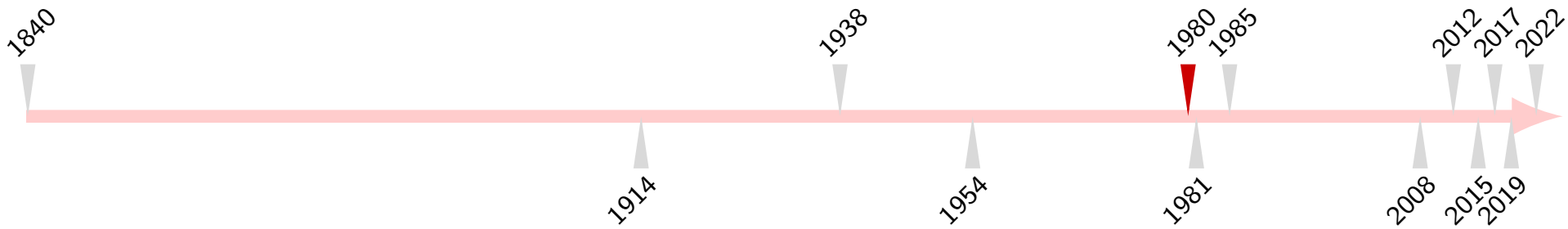
- 1840: Babbage and Lovelace conceptualize the first floating-point.
- 1914: Leonardo T. Quevedo builds an electromechanical calculator with floating-point.
- 1938: Konrad Zuse **Z1**, the first programmable mechanical computer with floating-point arithmetic.
- 1954: IBM 704, the first mass-produced computer with floating-point hardware.
- 1980: Intel introduces the 8087 co-processor.
- 1981: **Kulisch**: "Computer Arithmetic in Theory and Practice"
- 1985: **IEEE754**-85 standard.
- 2008: IEEE754-08 introduces half-precision and FMA
- 2012: AlexNet sparks the AI revolution with its CNN structure.
- 2015: Introduction of bfloat16, optimizing neural network computations.
- 2017: Introduction of the **Posit** format, a new take on floating-point representation.
- 2019: IEEE754 standard evolves to fix errors
- 2022: **LLM revolution**, chatgpt, MX format (e2m2, e4m3, e5m2), tensor cores

Historical Background of Floating-Point Arithmetic



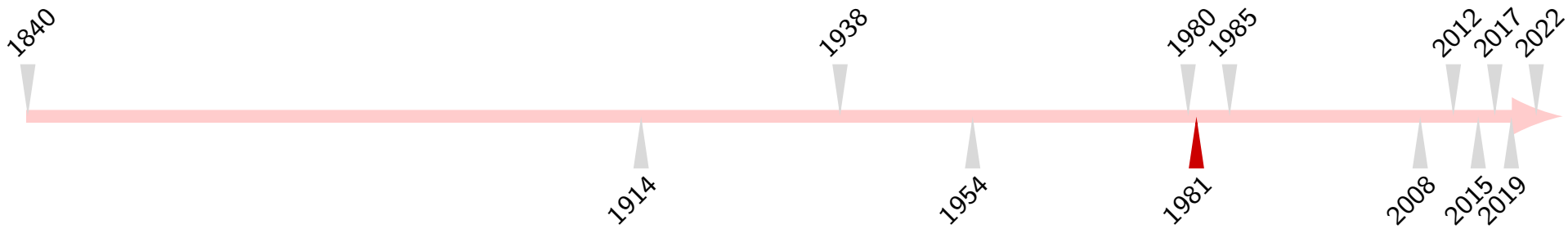
- 1840: Babbage and Lovelace conceptualize the first floating-point.
- 1914: Leonardo T. Quevedo builds an electromechanical calculator with floating-point.
- 1938: Konrad Zuse **Z1**, the first programmable mechanical computer with floating-point arithmetic.
- 1954: IBM 704, the first mass-produced computer with floating-point hardware.
- 1980: Intel introduces the 8087 co-processor.
- 1981: **Kulisch**: "Computer Arithmetic in Theory and Practice"
- 1985: **IEEE754**-85 standard.
- 2008: IEEE754-08 introduces half-precision and FMA
- 2012: AlexNet sparks the AI revolution with its CNN structure.
- 2015: Introduction of bfloat16, optimizing neural network computations.
- 2017: Introduction of the **Posit** format, a new take on floating-point representation.
- 2019: IEEE754 standard evolves to fix errors
- 2022: **LLM revolution**, chatgpt, MX format (e2m2, e4m3, e5m2), tensor cores

Historical Background of Floating-Point Arithmetic



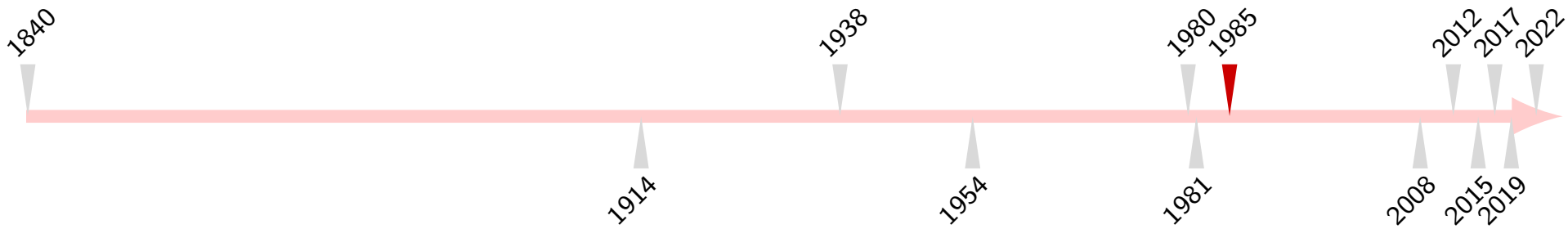
- 1840: Babbage and Lovelace conceptualize the first floating-point.
- 1914: Leonardo T. Quevedo builds an electromechanical calculator with floating-point.
- 1938: Konrad Zuse **Z1**, the first programmable mechanical computer with floating-point arithmetic.
- 1954: IBM 704, the first mass-produced computer with floating-point hardware.
- 1980: Intel introduces the 8087 co-processor.
- 1981: **Kulisch**: "Computer Arithmetic in Theory and Practice"
- 1985: **IEEE754**-85 standard.
- 2008: IEEE754-08 introduces half-precision and FMA
- 2012: AlexNet sparks the AI revolution with its CNN structure.
- 2015: Introduction of bfloat16, optimizing neural network computations.
- 2017: Introduction of the **Posit** format, a new take on floating-point representation.
- 2019: IEEE754 standard evolves to fix errors
- 2022: **LLM revolution**, chatgpt, MX format (e2m2, e4m3, e5m2), tensor cores

Historical Background of Floating-Point Arithmetic



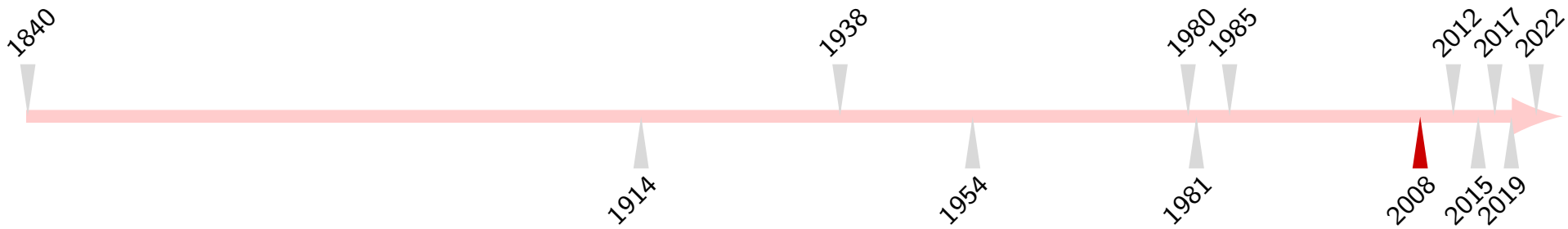
- 1840: Babbage and Lovelace conceptualize the first floating-point.
- 1914: Leonardo T. Quevedo builds an electromechanical calculator with floating-point.
- 1938: Konrad Zuse **Z1**, the first programmable mechanical computer with floating-point arithmetic.
- 1954: IBM 704, the first mass-produced computer with floating-point hardware.
- 1980: Intel introduces the 8087 co-processor.
- 1981: **Kulisch**: "Computer Arithmetic in Theory and Practice"
- 1985: **IEEE754**-85 standard.
- 2008: IEEE754-08 introduces half-precision and FMA
- 2012: AlexNet sparks the AI revolution with its CNN structure.
- 2015: Introduction of bfloat16, optimizing neural network computations.
- 2017: Introduction of the **Posit** format, a new take on floating-point representation.
- 2019: IEEE754 standard evolves to fix errors
- 2022: **LLM revolution**, chatgpt, MX format (e2m2, e4m3, e5m2), tensor cores

Historical Background of Floating-Point Arithmetic



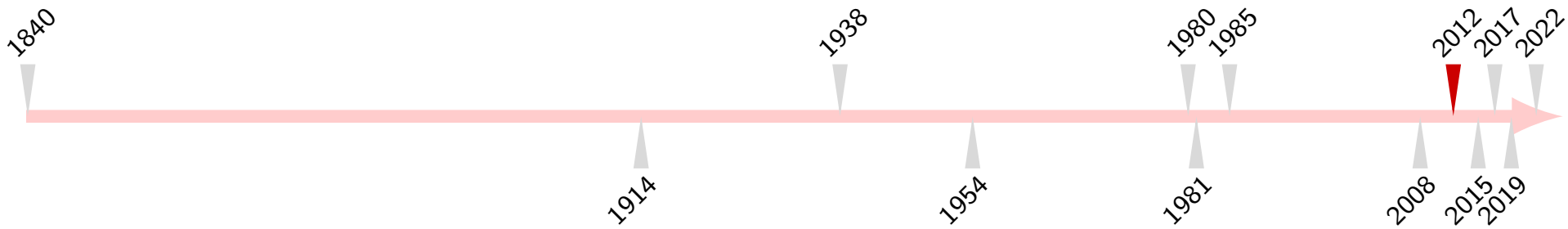
- 1840: Babbage and Lovelace conceptualize the first floating-point.
- 1914: Leonardo T. Quevedo builds an electromechanical calculator with floating-point.
- 1938: Konrad Zuse **Z1**, the first programmable mechanical computer with floating-point arithmetic.
- 1954: IBM 704, the first mass-produced computer with floating-point hardware.
- 1980: Intel introduces the 8087 co-processor.
- 1981: **Kulisch**: "Computer Arithmetic in Theory and Practice"
- 1985: **IEEE754**-85 standard.
- 2008: IEEE754-08 introduces half-precision and FMA
- 2012: AlexNet sparks the AI revolution with its CNN structure.
- 2015: Introduction of bfloat16, optimizing neural network computations.
- 2017: Introduction of the **Posit** format, a new take on floating-point representation.
- 2019: IEEE754 standard evolves to fix errors
- 2022: **LLM revolution**, chatgpt, MX format (e2m2, e4m3, e5m2), tensor cores

Historical Background of Floating-Point Arithmetic



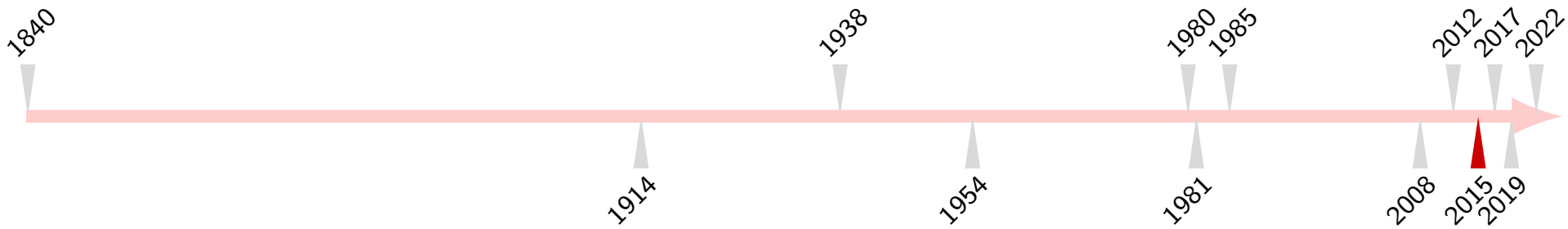
- 1840: Babbage and Lovelace conceptualize the first floating-point.
- 1914: Leonardo T. Quevedo builds an electromechanical calculator with floating-point.
- 1938: Konrad Zuse **Z1**, the first programmable mechanical computer with floating-point arithmetic.
- 1954: IBM 704, the first mass-produced computer with floating-point hardware.
- 1980: Intel introduces the 8087 co-processor.
- 1981: **Kulisch**: "Computer Arithmetic in Theory and Practice"
- 1985: **IEEE754**-85 standard.
- 2008: IEEE754-08 introduces half-precision and FMA
- 2012: AlexNet sparks the AI revolution with its CNN structure.
- 2015: Introduction of bfloat16, optimizing neural network computations.
- 2017: Introduction of the **Posit** format, a new take on floating-point representation.
- 2019: IEEE754 standard evolves to fix errors
- 2022: **LLM revolution**, chatgpt, MX format (e2m2, e4m3, e5m2), tensor cores

Historical Background of Floating-Point Arithmetic



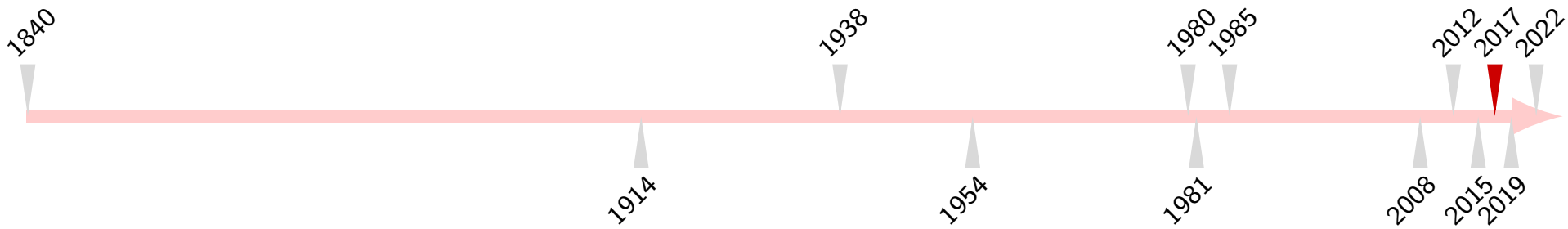
- 1840: Babbage and Lovelace conceptualize the first floating-point.
- 1914: Leonardo T. Quevedo builds an electromechanical calculator with floating-point.
- 1938: Konrad Zuse **Z1**, the first programmable mechanical computer with floating-point arithmetic.
- 1954: IBM 704, the first mass-produced computer with floating-point hardware.
- 1980: Intel introduces the 8087 co-processor.
- 1981: **Kulisch**: "Computer Arithmetic in Theory and Practice"
- 1985: **IEEE754**-85 standard.
- 2008: IEEE754-08 introduces half-precision and FMA
- 2012: AlexNet sparks the AI revolution with its CNN structure.
- 2015: Introduction of bfloat16, optimizing neural network computations.
- 2017: Introduction of the **Posit** format, a new take on floating-point representation.
- 2019: IEEE754 standard evolves to fix errors
- 2022: **LLM revolution**, chatgpt, MX format (e2m2, e4m3, e5m2), tensor cores

Historical Background of Floating-Point Arithmetic



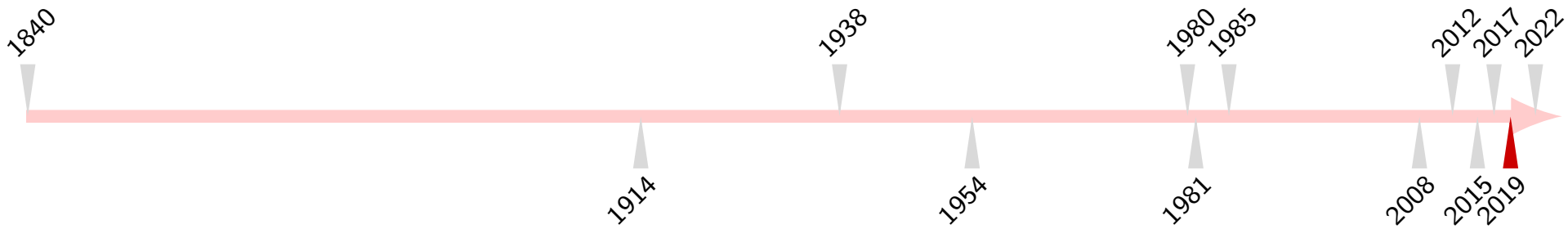
- 1840: Babbage and Lovelace conceptualize the first floating-point.
- 1914: Leonardo T. Quevedo builds an electromechanical calculator with floating-point.
- 1938: Konrad Zuse **Z1**, the first programmable mechanical computer with floating-point arithmetic.
- 1954: IBM 704, the first mass-produced computer with floating-point hardware.
- 1980: Intel introduces the 8087 co-processor.
- 1981: **Kulisch**: "Computer Arithmetic in Theory and Practice"
- 1985: **IEEE754**-85 standard.
- 2008: IEEE754-08 introduces half-precision and FMA
- 2012: AlexNet sparks the AI revolution with its CNN structure.
- 2015: Introduction of bfloat16, optimizing neural network computations.
- 2017: Introduction of the **Posit** format, a new take on floating-point representation.
- 2019: IEEE754 standard evolves to fix errors
- 2022: **LLM revolution**, chatgpt, MX format (e2m2, e4m3, e5m2), tensor cores

Historical Background of Floating-Point Arithmetic



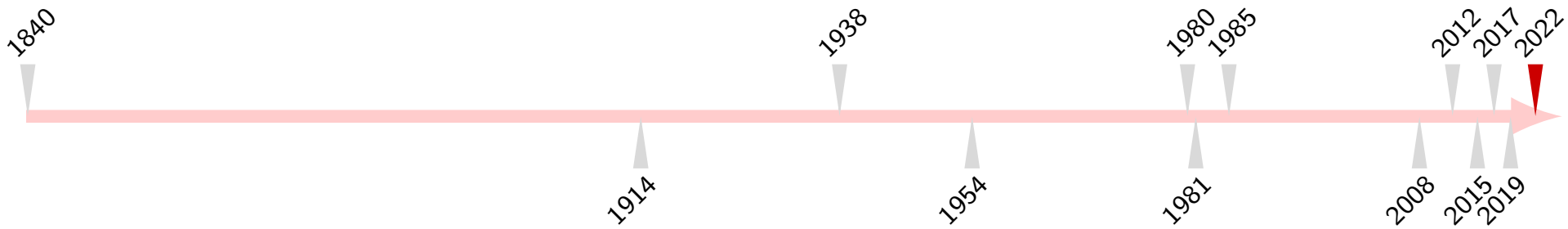
- 1840: Babbage and Lovelace conceptualize the first floating-point.
- 1914: Leonardo T. Quevedo builds an electromechanical calculator with floating-point.
- 1938: Konrad Zuse **Z1**, the first programmable mechanical computer with floating-point arithmetic.
- 1954: IBM 704, the first mass-produced computer with floating-point hardware.
- 1980: Intel introduces the 8087 co-processor.
- 1981: **Kulisch**: "Computer Arithmetic in Theory and Practice"
- 1985: **IEEE754**-85 standard.
- 2008: IEEE754-08 introduces half-precision and FMA
- 2012: AlexNet sparks the AI revolution with its CNN structure.
- 2015: Introduction of bfloat16, optimizing neural network computations.
- 2017: Introduction of the **Posit** format, a new take on floating-point representation.
- 2019: IEEE754 standard evolves to fix errors
- 2022: **LLM revolution**, chatgpt, MX format (e2m2, e4m3, e5m2), tensor cores

Historical Background of Floating-Point Arithmetic



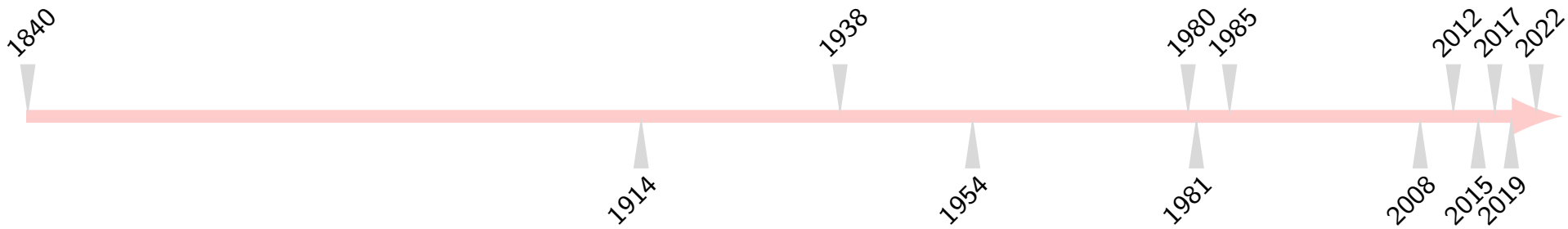
- 1840: Babbage and Lovelace conceptualize the first floating-point.
- 1914: Leonardo T. Quevedo builds an electromechanical calculator with floating-point.
- 1938: Konrad Zuse **Z1**, the first programmable mechanical computer with floating-point arithmetic.
- 1954: IBM 704, the first mass-produced computer with floating-point hardware.
- 1980: Intel introduces the 8087 co-processor.
- 1981: **Kulisch**: "Computer Arithmetic in Theory and Practice"
- 1985: **IEEE754**-85 standard.
- 2008: IEEE754-08 introduces half-precision and FMA
- 2012: AlexNet sparks the AI revolution with its CNN structure.
- 2015: Introduction of bfloat16, optimizing neural network computations.
- 2017: Introduction of the **Posit** format, a new take on floating-point representation.
- 2019: IEEE754 standard evolves to fix errors
- 2022: **LLM revolution**, chatgpt, MX format (e2m2, e4m3, e5m2), tensor cores

Historical Background of Floating-Point Arithmetic



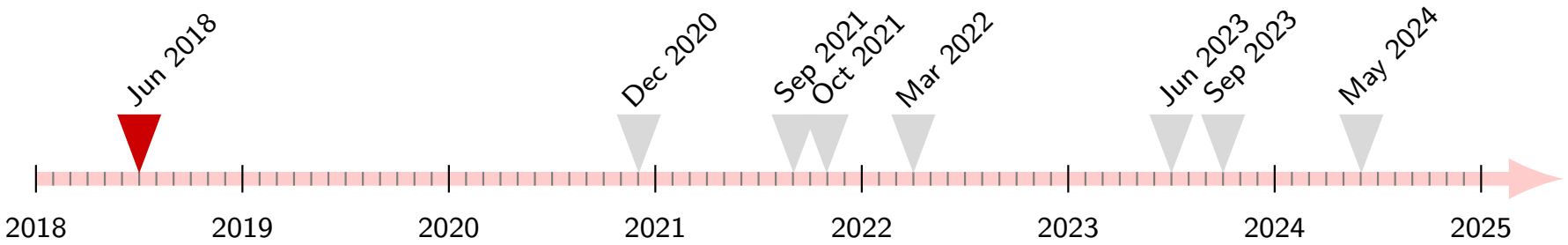
- 1840: Babbage and Lovelace conceptualize the first floating-point.
- 1914: Leonardo T. Quevedo builds an electromechanical calculator with floating-point.
- 1938: Konrad Zuse **Z1**, the first programmable mechanical computer with floating-point arithmetic.
- 1954: IBM 704, the first mass-produced computer with floating-point hardware.
- 1980: Intel introduces the 8087 co-processor.
- 1981: **Kulisch**: "Computer Arithmetic in Theory and Practice"
- 1985: **IEEE754**-85 standard.
- 2008: IEEE754-08 introduces half-precision and FMA
- 2012: AlexNet sparks the AI revolution with its CNN structure.
- 2015: Introduction of bfloat16, optimizing neural network computations.
- 2017: Introduction of the **Posit** format, a new take on floating-point representation.
- 2019: IEEE754 standard evolves to fix errors
- 2022: **LLM revolution**, chatgpt, MX format (e2m2, e4m3, e5m2), tensor cores

Historical Background of Floating-Point Arithmetic



- 1840: Babbage and Lovelace conceptualize the first floating-point.
- 1914: Leonardo T. Quevedo builds an electromechanical calculator with floating-point.
- 1938: Konrad Zuse **Z1**, the first programmable mechanical computer with floating-point arithmetic.
- 1954: IBM 704, the first mass-produced computer with floating-point hardware.
- 1980: Intel introduces the 8087 co-processor.
- 1981: **Kulisch**: "Computer Arithmetic in Theory and Practice"
- 1985: **IEEE754**-85 standard.
- 2008: IEEE754-08 introduces half-precision and FMA
- 2012: AlexNet sparks the AI revolution with its CNN structure.
- 2015: Introduction of bfloat16, optimizing neural network computations.
- 2017: Introduction of the **Posit** format, a new take on floating-point representation.
- 2019: IEEE754 standard evolves to fix errors
- 2022: **LLM revolution**, chatgpt, MX format (e2m2, e4m3, e5m2), tensor cores

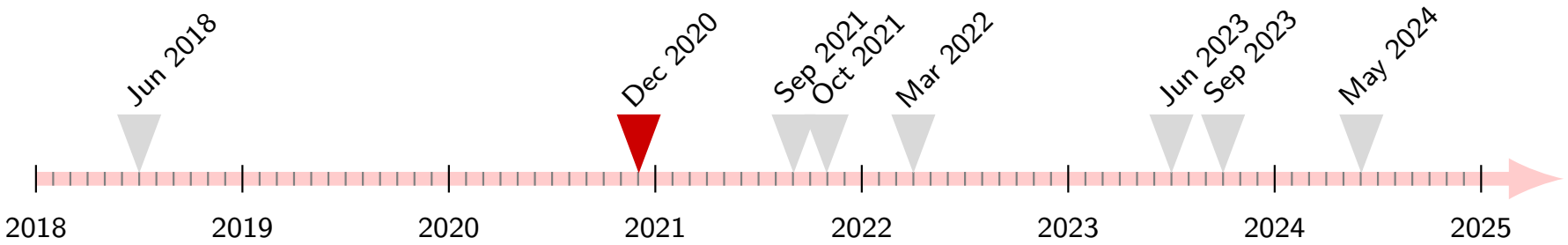
Timeline Zoom: "LLM Revolution"



1. Jun 2018: Posit draft evolves.
2. Dec 2020: MSFP at NeurIPS [RLZ+20].
3. Sep 2021: Unum-IV [Ser21].
4. Oct 2021: Tesla Cfloat8/Cfloat16 [TWS22].
5. Mar 2022: Posit draft becomes a standard.
6. Jun 2023: Bfloat16 with subnormals.
7. Jun 2023: MXFP formats MXINT8 [DRZE+23].
8. Jun 2023: OCP 8-bit FP spec.
9. Sep 2023: OCP announcement [RZM+23].
10. Sep 2023: IEEE P3109 report on 8-bit FP.
11. May 2024: PT-Float at ARITH 2024 [SLS+24].
12. May 2024: IEEE P3109: 28 new formats.

Approximate number of new float formats: 5

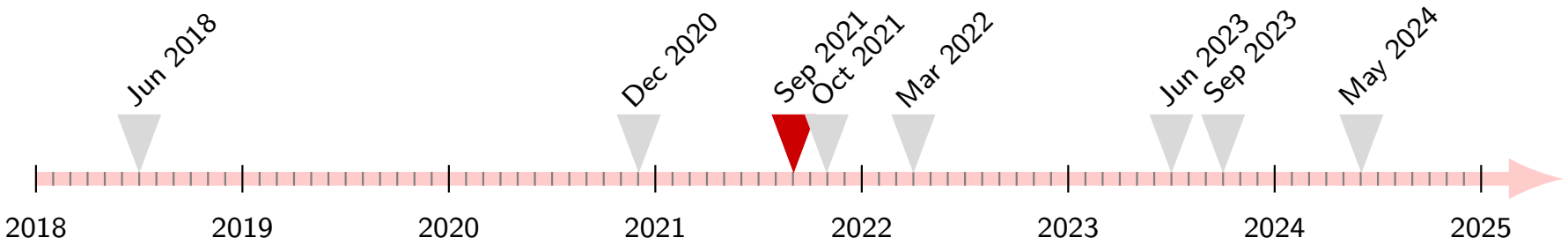
Timeline Zoom: "LLM Revolution"



1. Jun 2018: Posit draft evolves.
2. Dec 2020: MSFP at NeurIPS [RLZ+20].
3. Sep 2021: Unum-IV [Ser21].
4. Oct 2021: Tesla Cfloat8/Cfloat16 [TWS22].
5. Mar 2022: Posit draft becomes a standard.
6. Jun 2023: Bfloat16 with subnormals.
7. Jun 2023: MXFP formats MXINT8 [DRZE+23].
8. Jun 2023: OCP 8-bit FP spec.
9. Sep 2023: OCP announcement [RZM+23].
10. Sep 2023: IEEE P3109 report on 8-bit FP.
11. May 2024: PT-Float at ARITH 2024 [SLS+24].
12. May 2024: IEEE P3109: 28 new formats.

Approximate number of new float formats: 11

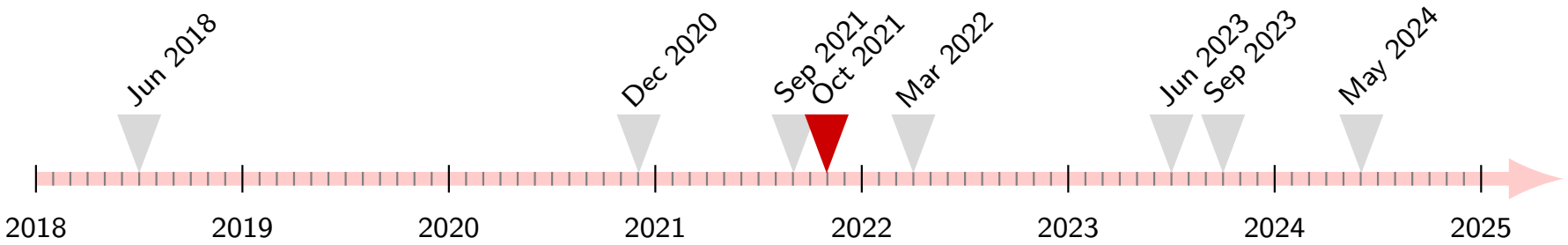
Timeline Zoom: "LLM Revolution"



1. Jun 2018: Posit draft evolves.
2. Dec 2020: MSFP at NeurIPS [RLZ+20].
3. Sep 2021: Unum-IV [Ser21].
4. Oct 2021: Tesla Cfloat8/Cfloat16 [TWS22].
5. Mar 2022: Posit draft becomes a standard.
6. Jun 2023: Bfloat16 with subnormals.
7. Jun 2023: MXFP formats MXINT8 [DRZE+23].
8. Jun 2023: OCP 8-bit FP spec.
9. Sep 2023: OCP announcement [RZM+23].
10. Sep 2023: IEEE P3109 report on 8-bit FP.
11. May 2024: PT-Float at ARITH 2024 [SLS+24].
12. May 2024: IEEE P3109: 28 new formats.

Approximate number of new float formats: 12

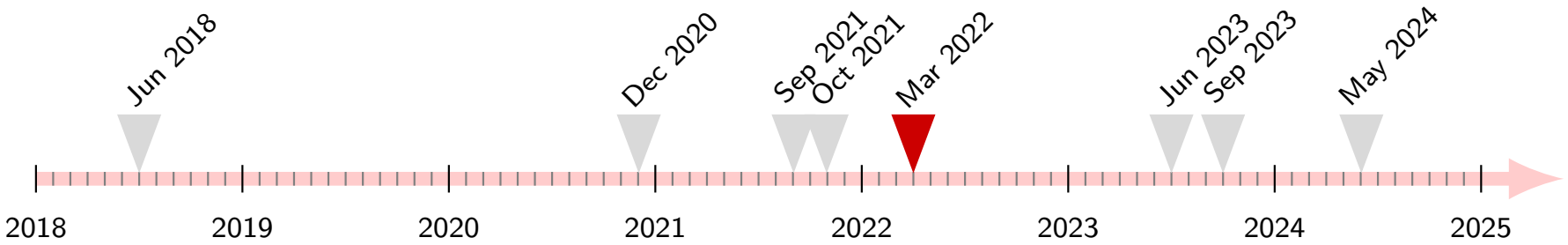
Timeline Zoom: "LLM Revolution"



1. Jun 2018: Posit draft evolves.
2. Dec 2020: MSFP at NeurIPS [RLZ+20].
3. Sep 2021: Unum-IV [Ser21].
4. Oct 2021: Tesla Cfloat8/Cfloat16 [TWS22].
5. Mar 2022: Posit draft becomes a standard.
6. Jun 2023: Bfloat16 with subnormals.
7. Jun 2023: MXFP formats MXINT8 [DRZE+23].
8. Jun 2023: OCP 8-bit FP spec.
9. Sep 2023: OCP announcement [RZM+23].
10. Sep 2023: IEEE P3109 report on 8-bit FP.
11. May 2024: PT-Float at ARITH 2024 [SLS+24].
12. May 2024: IEEE P3109: 28 new formats.

Approximate number of new float formats: 14

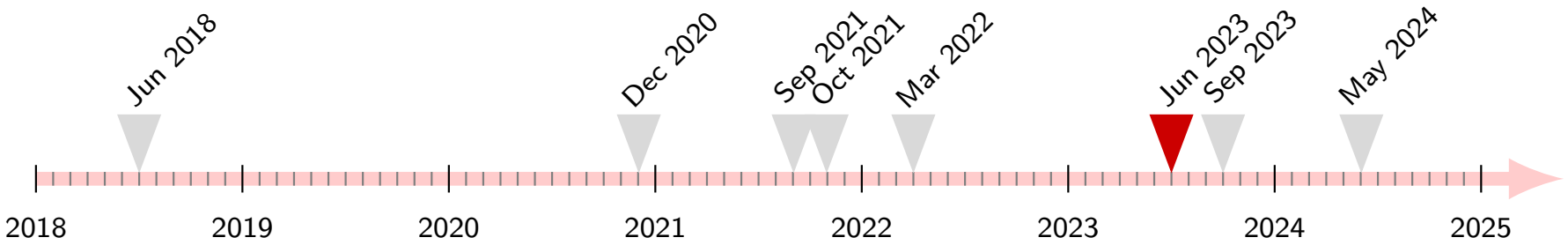
Timeline Zoom: "LLM Revolution"



1. Jun 2018: Posit draft evolves.
2. Dec 2020: MSFP at NeurIPS [RLZ+20].
3. Sep 2021: Unum-IV [Ser21].
4. Oct 2021: Tesla Cfloat8/Cfloat16 [TWS22].
5. Mar 2022: Posit draft becomes a standard.
6. Jun 2023: Bfloat16 with subnormals.
7. Jun 2023: MXFP formats MXINT8 [DRZE+23].
8. Jun 2023: OCP 8-bit FP spec.
9. Sep 2023: OCP announcement [RZM+23].
10. Sep 2023: IEEE P3109 report on 8-bit FP.
11. May 2024: PT-Float at ARITH 2024 [SLS+24].
12. May 2024: IEEE P3109: 28 new formats.

Approximate number of new float formats: 19

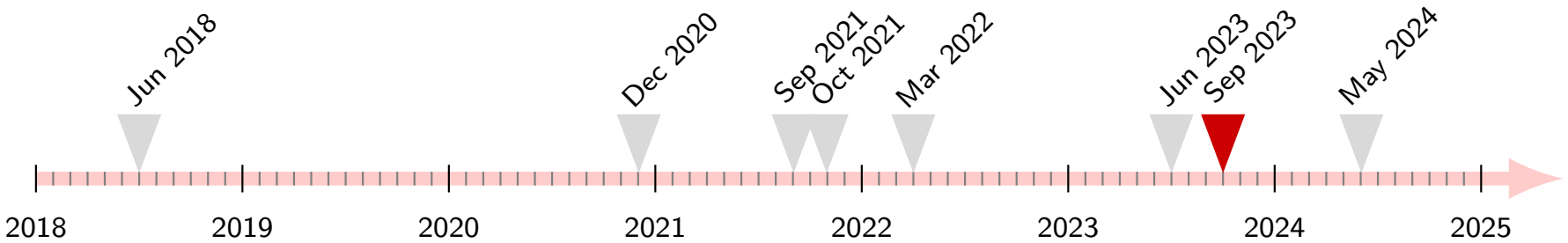
Timeline Zoom: "LLM Revolution"



1. Jun 2018: Posit draft evolves.
2. Dec 2020: MSFP at NeurIPS [RLZ+20].
3. Sep 2021: Unum-IV [Ser21].
4. Oct 2021: Tesla Cfloat8/Cfloat16 [TWS22].
5. Mar 2022: Posit draft becomes a standard.
6. Jun 2023: Bfloat16 with subnormals.
7. Jun 2023: MXFP formats MXINT8 [DRZE+23].
8. Jun 2023: OCP 8-bit FP spec.
9. Sep 2023: OCP announcement [RZM+23].
10. Sep 2023: IEEE P3109 report on 8-bit FP.
11. May 2024: PT-Float at ARITH 2024 [SLS+24].
12. May 2024: IEEE P3109: 28 new formats.

Approximate number of new float formats: 32

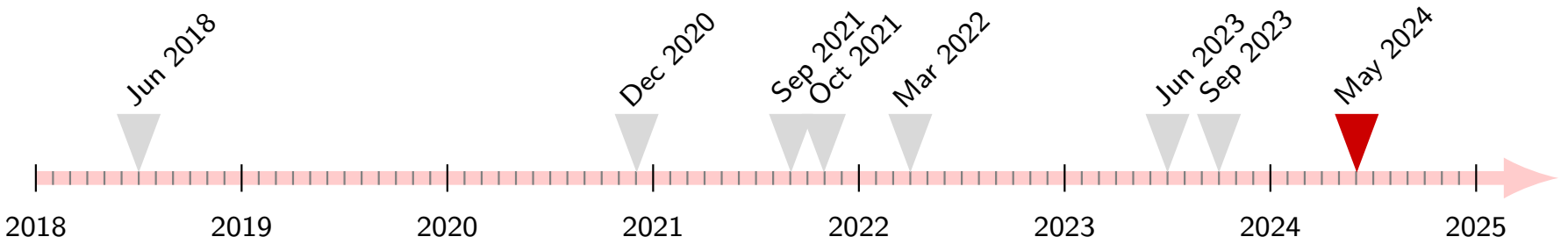
Timeline Zoom: "LLM Revolution"



1. Jun 2018: Posit draft evolves.
2. Dec 2020: MSFP at NeurIPS [RLZ+20].
3. Sep 2021: Unum-IV [Ser21].
4. Oct 2021: Tesla Cfloat8/Cfloat16 [TWS22].
5. Mar 2022: Posit draft becomes a standard.
6. Jun 2023: Bfloat16 with subnormals.
7. Jun 2023: MXFP formats MXINT8 [DRZE+23].
8. Jun 2023: OCP 8-bit FP spec.
9. Sep 2023: OCP announcement [RZM+23].
10. Sep 2023: IEEE P3109 report on 8-bit FP.
11. May 2024: PT-Float at ARITH 2024 [SLS+24].
12. May 2024: IEEE P3109: 28 new formats.

Approximate number of new float formats: 32

Timeline Zoom: "LLM Revolution"



1. Jun 2018: Posit draft evolves.
2. Dec 2020: MSFP at NeurIPS [RLZ+20].
3. Sep 2021: Unum-IV [Ser21].
4. Oct 2021: Tesla Cfloat8/Cfloat16 [TWS22].
5. Mar 2022: Posit draft becomes a standard.
6. Jun 2023: Bfloat16 with subnormals.
7. Jun 2023: MXFP formats MXINT8 [DRZE+23].
8. Jun 2023: OCP 8-bit FP spec.
9. Sep 2023: OCP announcement [RZM+23].
10. Sep 2023: IEEE P3109 report on 8-bit FP.
11. May 2024: PT-Float at ARITH 2024 [SLS+24].
12. May 2024: IEEE P3109: 28 new formats.

Approximate number of new float formats: **53**

Timeline Zoom: "LLM Revolution"

Modern history emphasizes "Arithmetic wall"

The "LLM revolution" introduces dozens of formats. SOTA constantly evolves and standards now specify **meta-formats**.

Another concern addressed by this thesis

This thesis tackles this issue through **format-agnosticism**. Contributions are **succesfully re-evaluated** using recent SOTA.

Timeline Zoom: "LLM Revolution"

Modern history emphasizes "Arithmetic wall"

The "LLM revolution" introduces dozens of formats. SOTA constantly evolves and standards now specify **meta-formats**.

Another concern addressed by this thesis

This thesis tackles this issue through **format-agnosticism**. Contributions are **succesfully re-evaluated** using recent SOTA.

Background: Kulisch Accumulation

- **Exact accumulator**.
- Fixed-point circuitry ⇒ **One bit per binade**.
- **Easy pipelining:** One branch accumulation.
- Shift in place the addend.
- Postpones rounding.
- **Fused** summation and dot product.

Cost consideration

IEEE754-64 **exact** accumulation of product requires over **4000 bits**.
Never adopted by the standard.

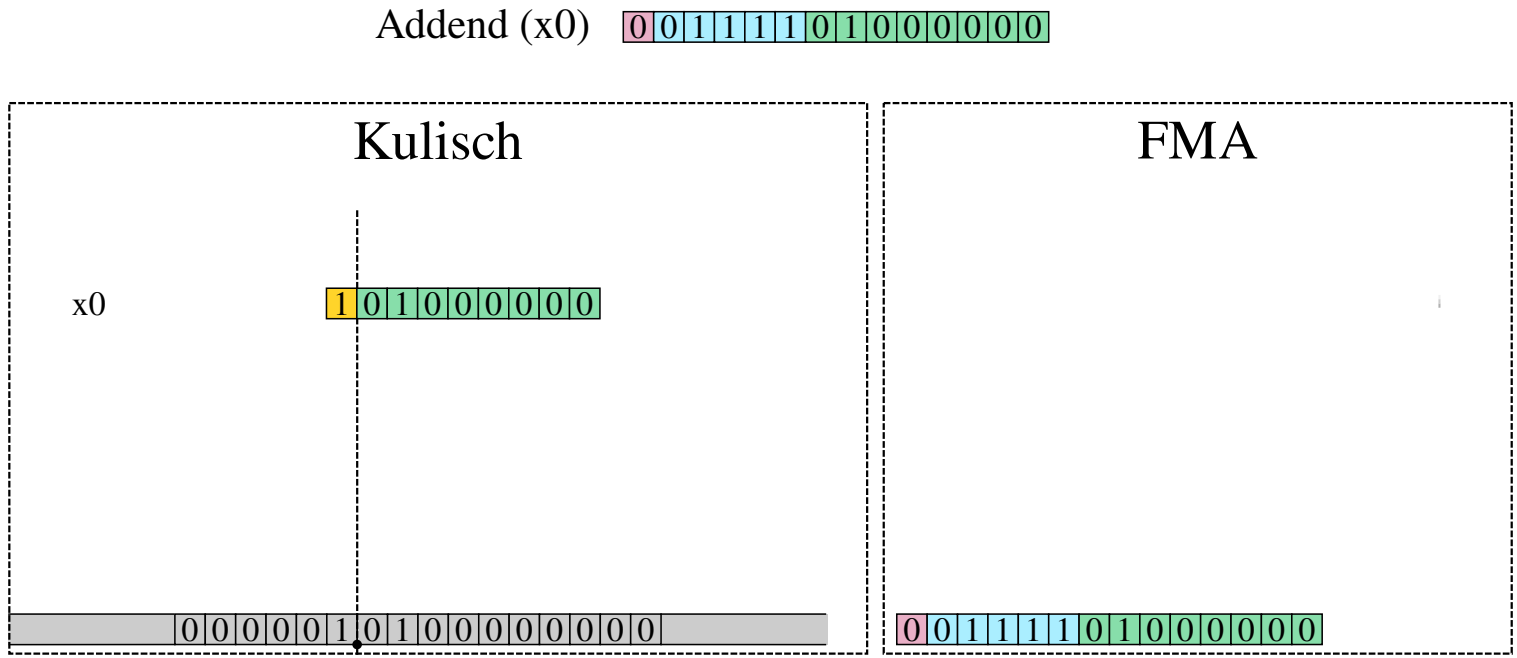
Background: Kulisch Accumulation

- **Exact accumulator**.
- Fixed-point circuitry ⇒ **One bit per binade**.
- **Easy pipelining:** One branch accumulation.
- Shift in place the addend.
- Postpones rounding.
- **Fused** summation and dot product.

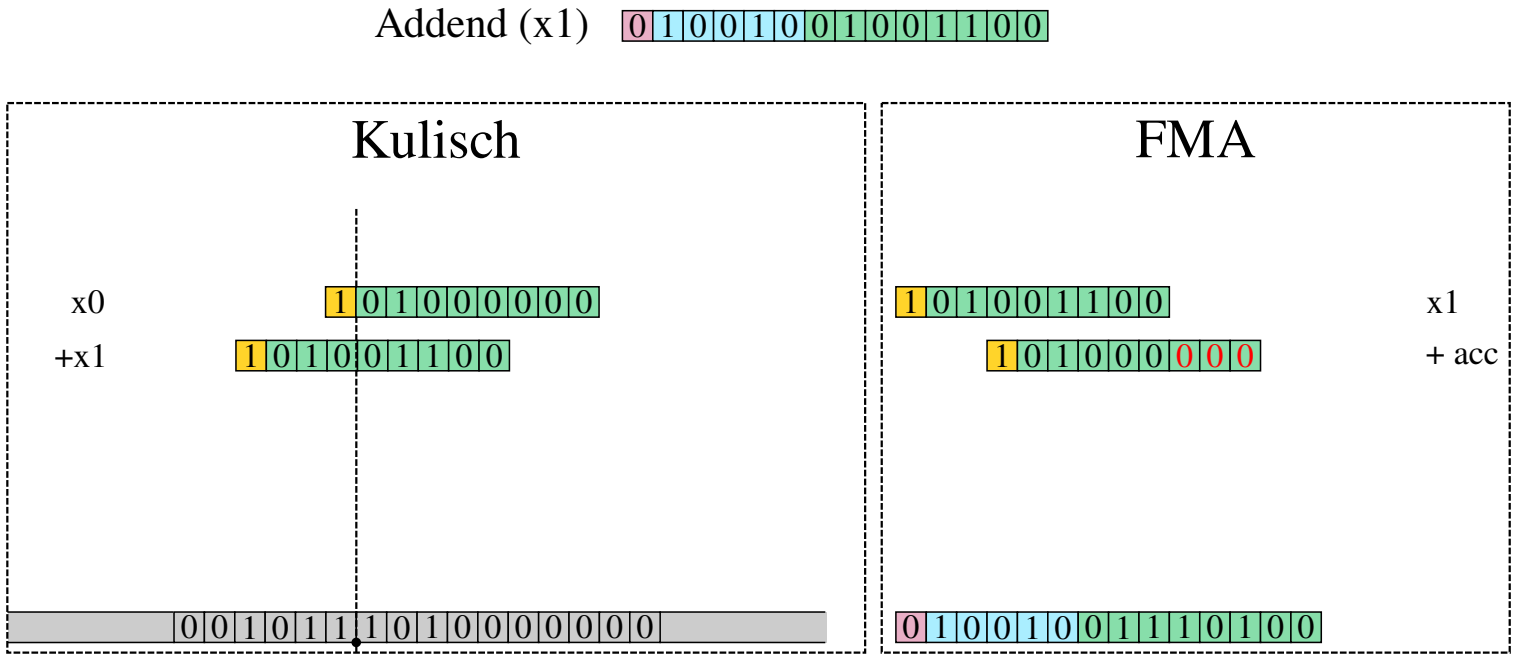
Cost consideration

IEEE754-64 **exact** accumulation of product requires over **4000 bits**.
Never adopted by the standard.

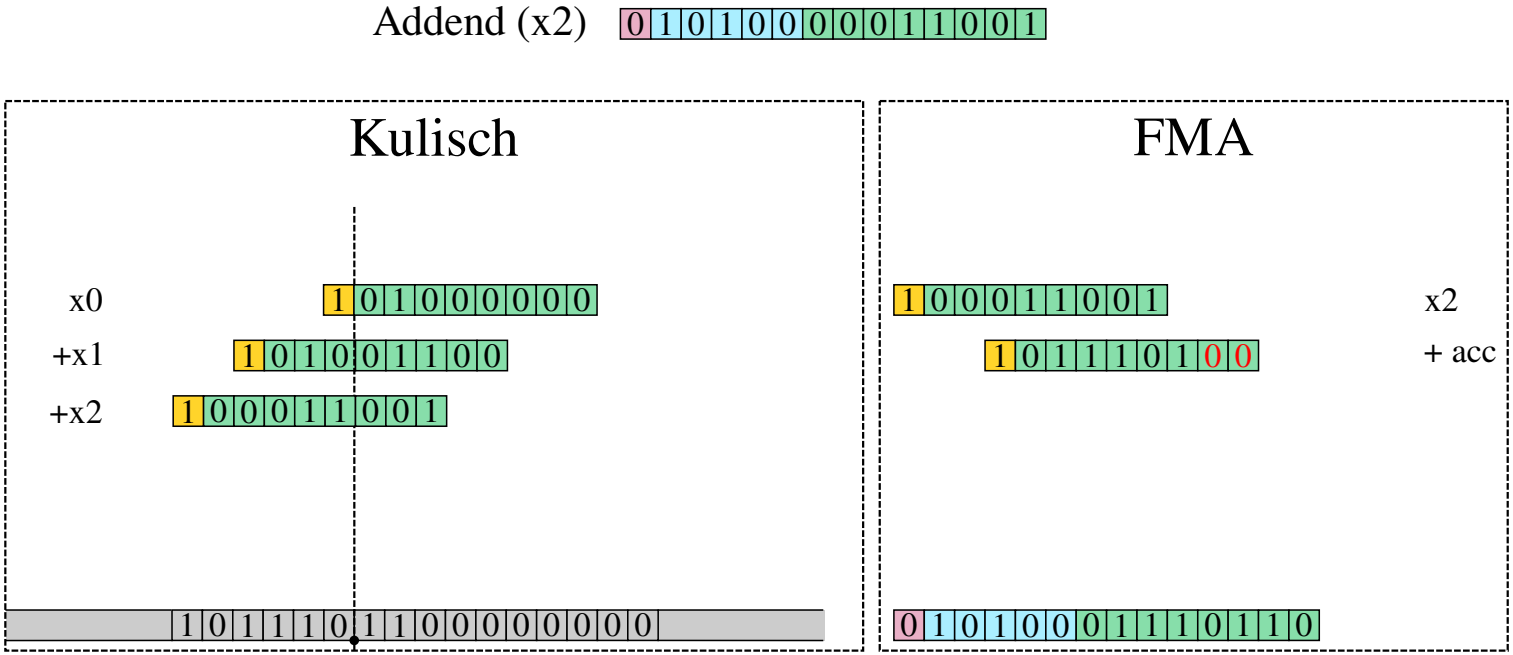
Comparison of Accumulation Methods



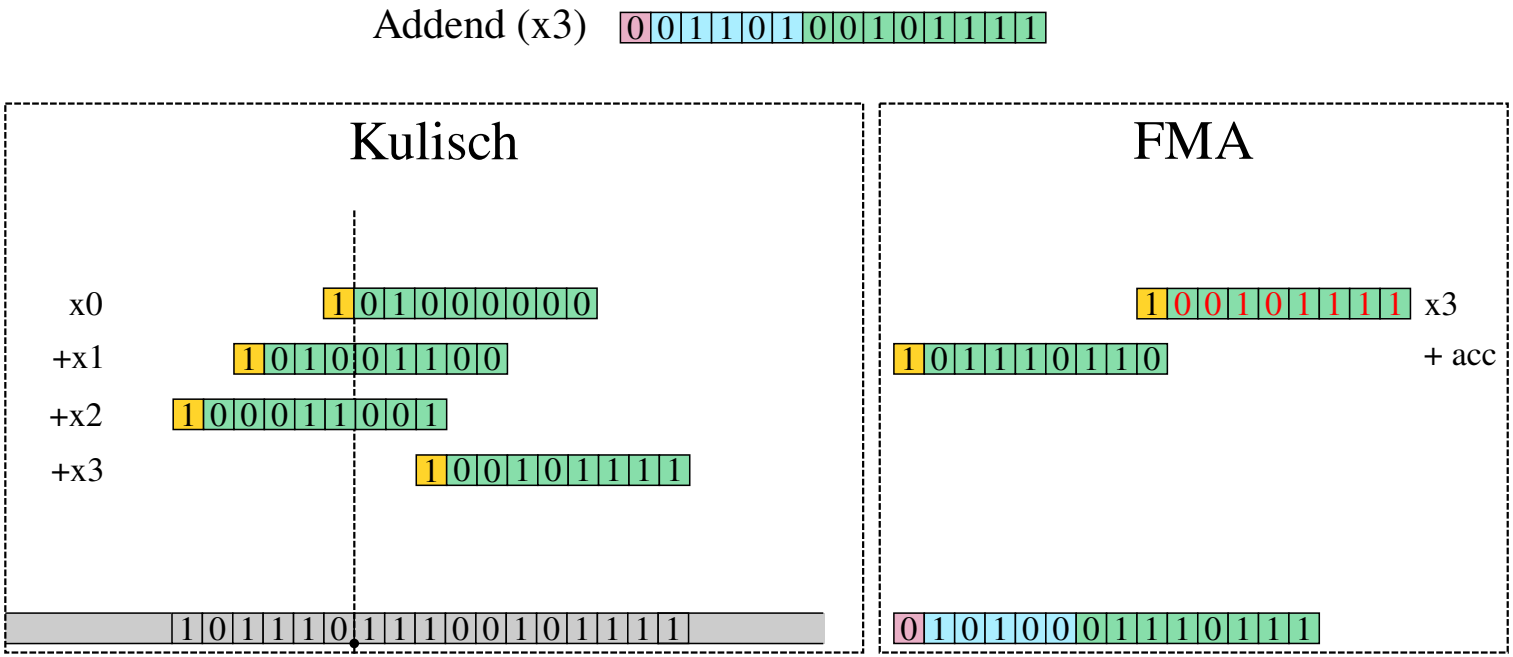
Comparison of Accumulation Methods



Comparison of Accumulation Methods



Comparison of Accumulation Methods



Comparison of Accumulation Methods

Results

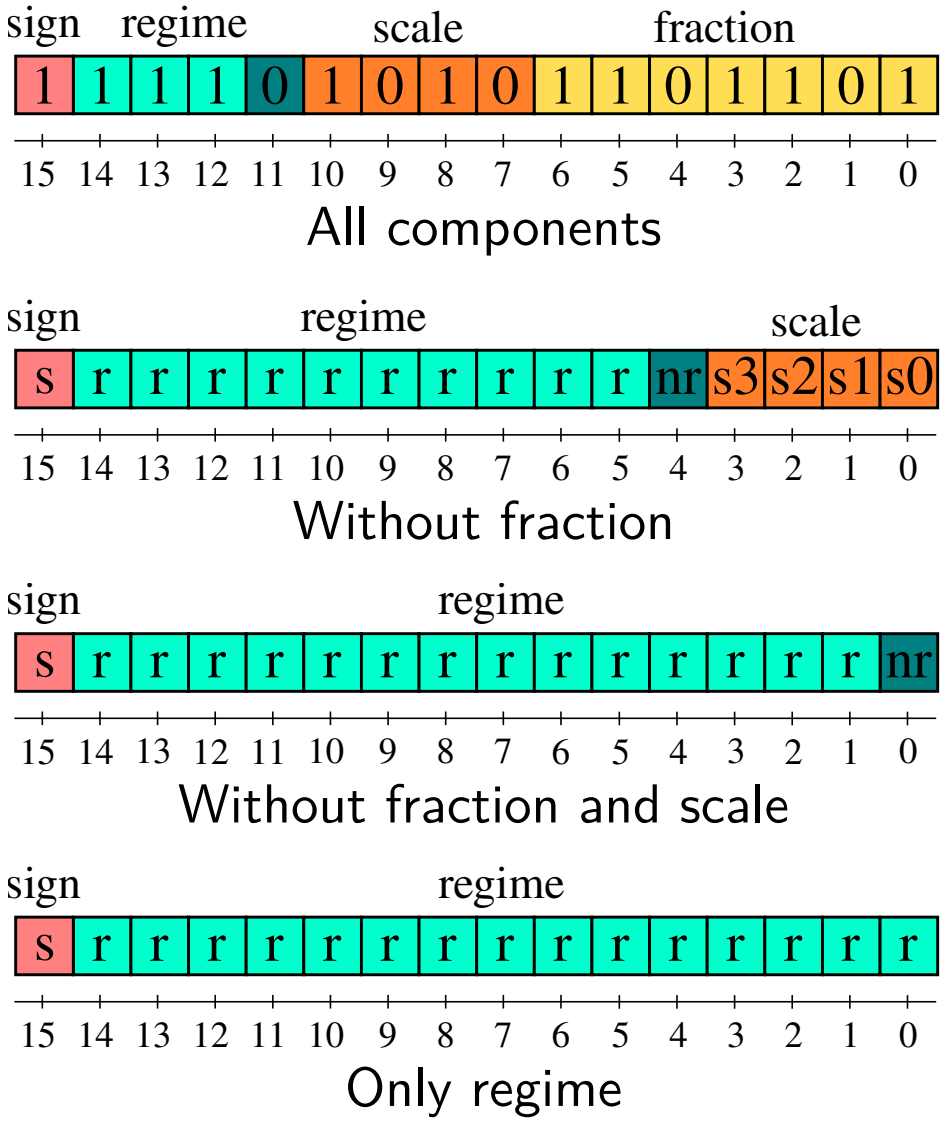
$$R_{\text{exact}} = 46.89794921875$$

$$R_{\text{FMA}} = 45.4375$$

$$R_{\text{Kulisch}} = 46.89794921875$$

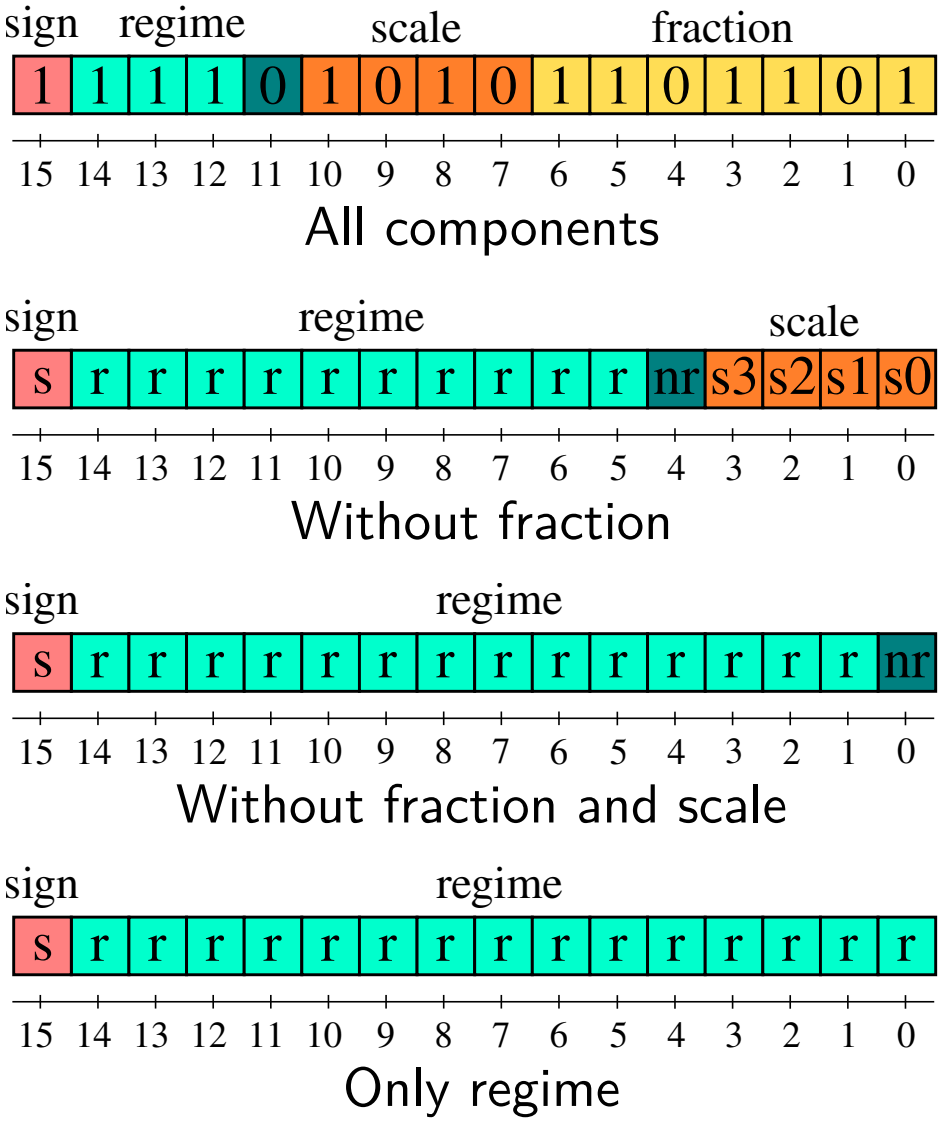
Background: Posit format

- **"Tapered-Precision"** Posit$\langle N, es \rangle$.
- N the bitwidth, es exponent size.
- **Variable-Lenght** fields.
- **Golomb Rice** regime (r).
- **Regime** is super-exponent
- **Quire:** Kulisch in standard.
- **Claim** to replace *IEEE754*
- **1** NaN
- **Dynamic Range** enhanced
- **Precision around one** enhanced



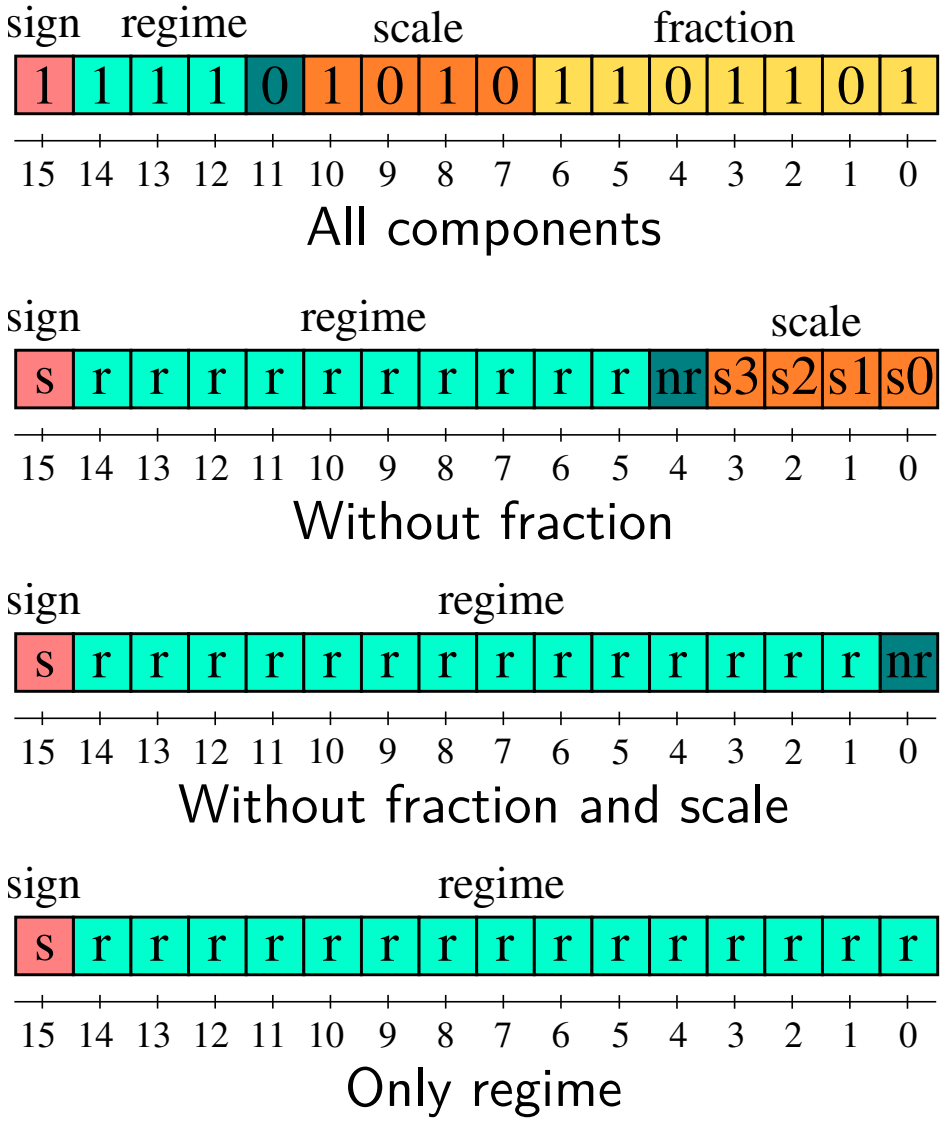
Background: Posit format

- **"Tapered-Precision"** Posit$\langle N, es \rangle$.
- N the bitwidth, es exponent size.
- **Variable-Lenght** fields.
- **Golomb Rice** regime (r).
- **Regime** is super-exponent
- **Quire:** Kulisch in standard.
- **Claim** to replace *IEEE754*
- **1** NaN
- **Dynamic Range** enhanced
- **Precision around one** enhanced



Background: Posit format

- **"Tapered-Precision"** Posit$\langle N, es \rangle$.
- N the bitwidth, es exponent size.
- **Variable-Lenght** fields.
- **Golomb Rice** regime (r).
- **Regime** is super-exponent
- **Quire:** Kulisch in standard.
- **Claim** to replace *IEEE754*
- **1** NaN
- **Dynamic Range** enhanced
- **Precision around one** enhanced



Current Progress

"Accelerating Deep Learning Inference with the Posit Number System"

Posit AI acceleration

2.1 Executive Summary and Motivations

2.2 Posit Operator Framework (POF)

2.3 Machine Learning Evaluation

2.4 Conclusions

Executive Summary

Nascent Nature

The Posit format is emerging, with **limited state-of-the-art** implementations that are often **closed-source**.

Objective

To evaluate Posit circuitry features and standards with pipelined operators in an **end-to-end** environment. Complete **Simulation-based** and **Operator-level** results.

Methodology

Employing a **Co-Design Accelerator Approach** on a cutting-edge POWER9 and CAPI2 platform to evaluate Posit logic in FPGA for **AI image classification**.

Executive Summary

Nascent Nature

The Posit format is emerging, with **limited state-of-the-art** implementations that are often **closed-source**.

Objective

To evaluate Posit circuitry features and standards with pipelined operators in an **end-to-end** environment. Complete **Simulation-based** and **Operator-level** results.

Methodology

Employing a **Co-Design Accelerator Approach** on a cutting-edge POWER9 and CAPI2 platform to evaluate Posit logic in FPGA for **AI image classification**.

Executive Summary

Nascent Nature

The Posit format is emerging, with **limited state-of-the-art** implementations that are often **closed-source**.

Objective

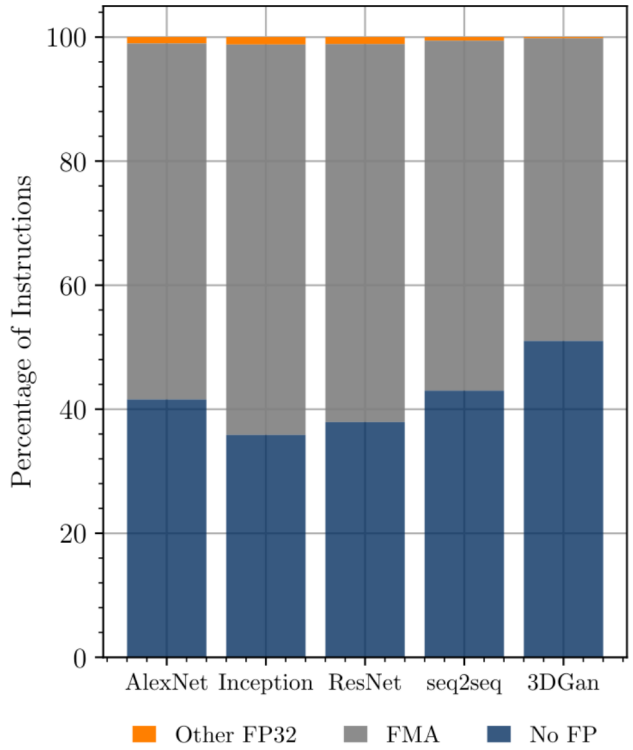
To evaluate Posit circuitry features and standards with pipelined operators in an **end-to-end** environment. Complete **Simulation-based** and **Operator-level** results.

Methodology

Employing a **Co-Design Accelerator Approach** on a cutting-edge POWER9 and CAPI2 platform to evaluate Posit logic in FPGA for **AI image classification**.

Error Accumulation Mitigation

- $\approx 50\%$ of CPU instructions are FMAs [ORAK$^+$20].
- Conventional arithmetic loses accuracy as errors accumulate.

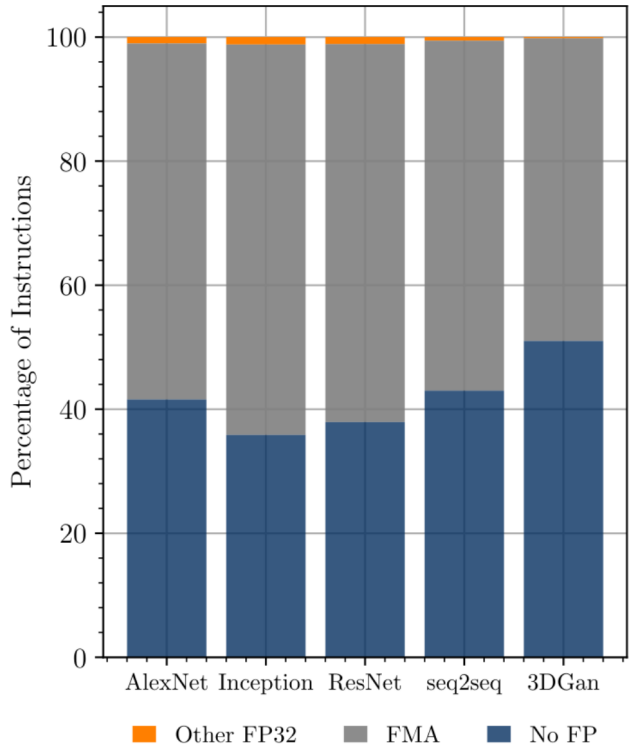


Instruction distributions for neural networks.

[ORAK$^+$20] Osorio Ríos et al., "Evaluating Mixed-Precision Arithmetic for 3D Generative Adversarial Networks to Simulate High Energy Physics Detectors" (2020).

Error Accumulation Mitigation

- $\approx 50\%$ of CPU instructions are FMAs [ORAK$^+$20].
- Conventional arithmetic loses accuracy as errors accumulate.



Instruction distributions for neural networks.

[ORAK$^+$20] Osorio Ríos et al., "Evaluating Mixed-Precision Arithmetic for 3D Generative Adversarial Networks to Simulate High Energy Physics Detectors" (2020).

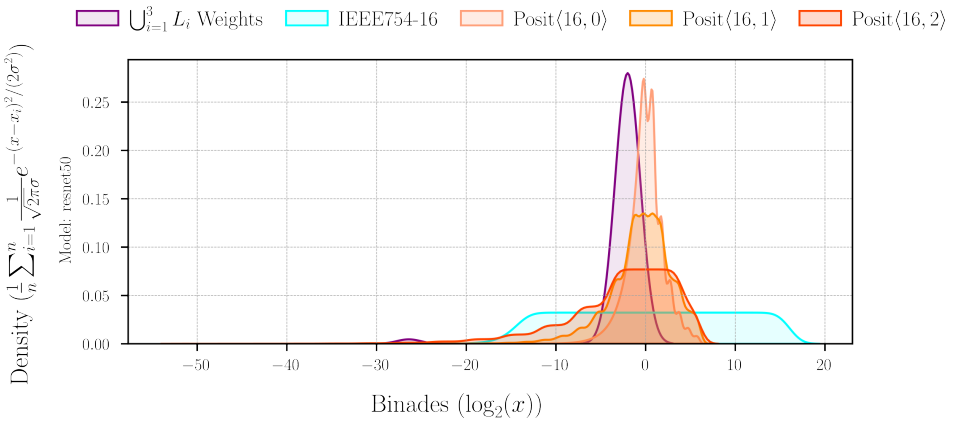
Error Accumulation Mitigation

Motivation

Posits prevent rounding errors with the quire feature.

Weight Distribution Optimizations

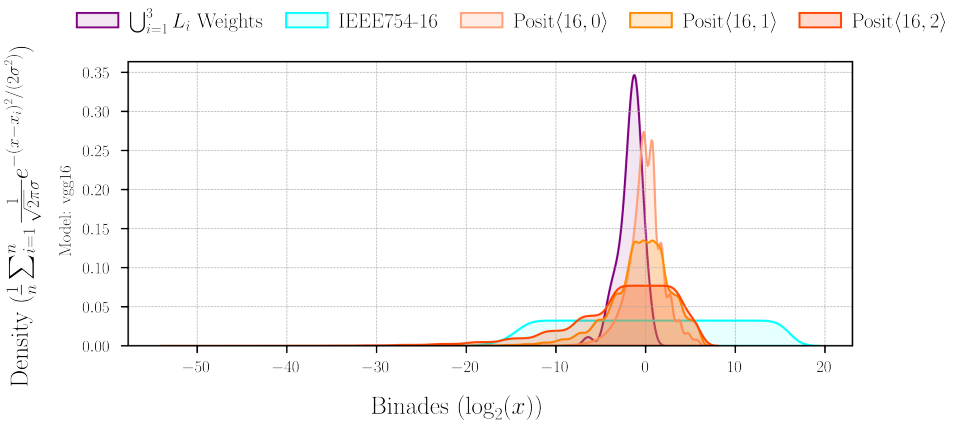
- Weight distribution of (model,datasets).
- This distribution centered around zero and does not spread.
- Matches the **"golden zone"**.



Distribution of VGG16 weights.

Weight Distribution Optimizations

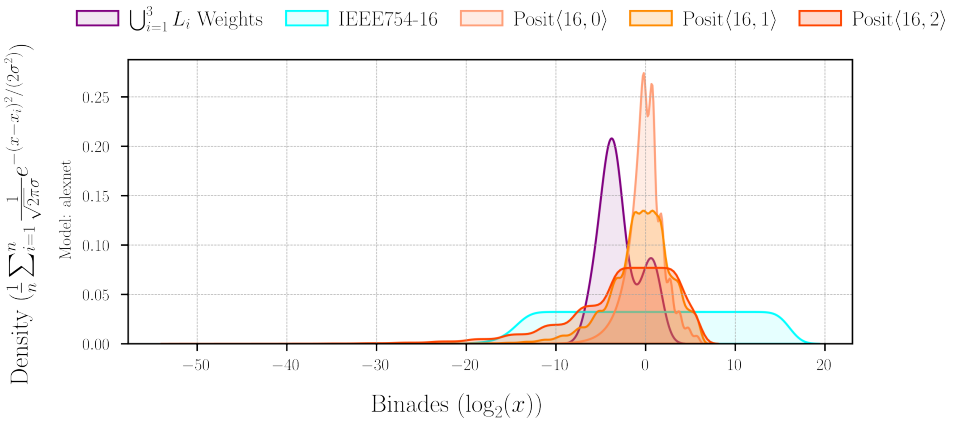
- Weight distribution of (model,datasets).
- This distribution centered around zero and does not spread.
- Matches the **"golden zone"**.



Distribution of Resnet50 weights.

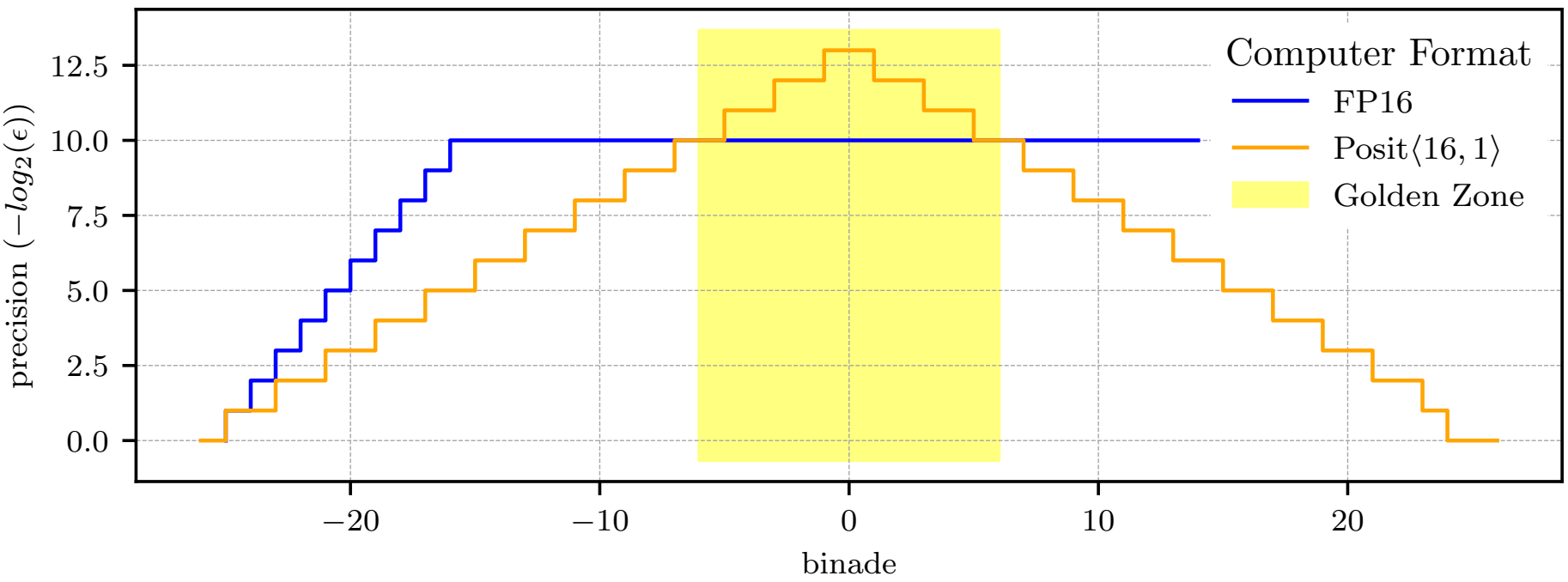
Weight Distribution Optimizations

- Weight distribution of (model,datasets).
- This distribution centered around zero and does not spread.
- Matches the **“golden zone”**.



Distribution of AlexNet weights.

Weight Distribution Optimizations



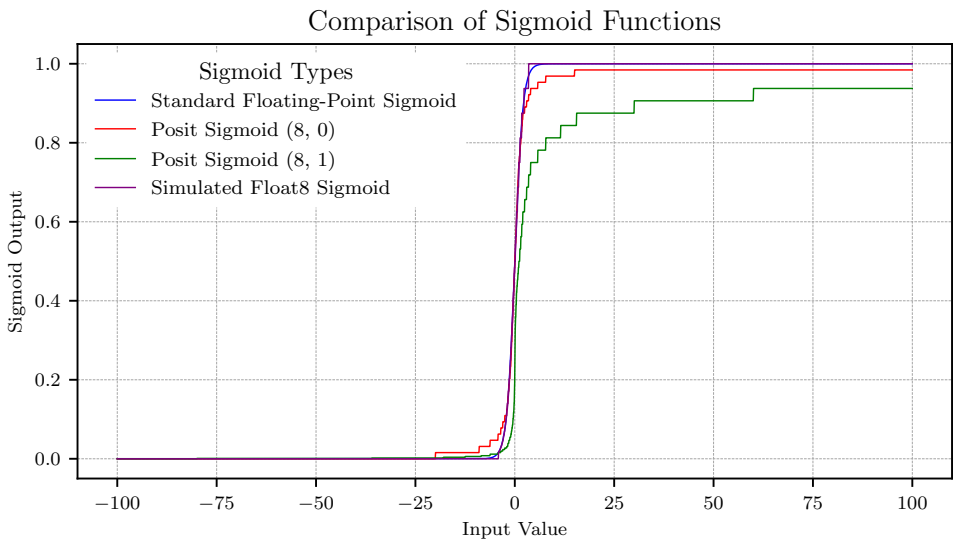
Comparison of precision per binade

Efficient Sigmoid Activation

Sigmoid introduce crucial non-linearity [DQZ18]:

$$\sigma(x) = \frac{1}{1 + e^{-x}}$$

- *IEEE754* have aproximations tricks (0x5f3759df)
- *Posits* defend with **nearly 0 cost** sigmoid approximation
- MSB bitflip followed by constant bitshift



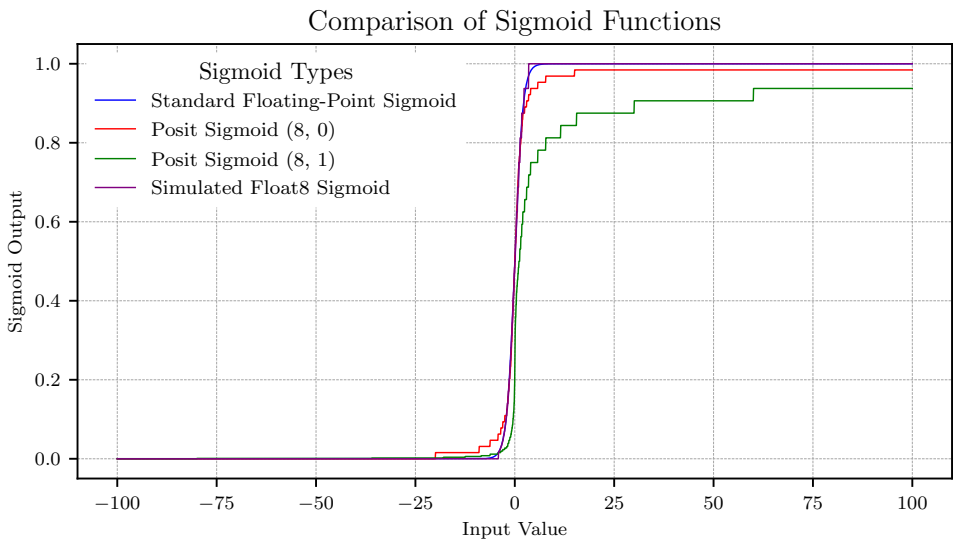
Sigmoid with 8-bit float types.

Efficient Sigmoid Activation

Sigmoid introduce crucial non-linearity [DQZ18]:

$$\sigma(x) = \frac{1}{1 + e^{-x}}$$

- *IEEE754* have aproximations tricks (0x5f3759df)
- *Posits* defend with **nearly 0 cost** sigmoid approximation
- MSB bitflip followed by constant bitshift



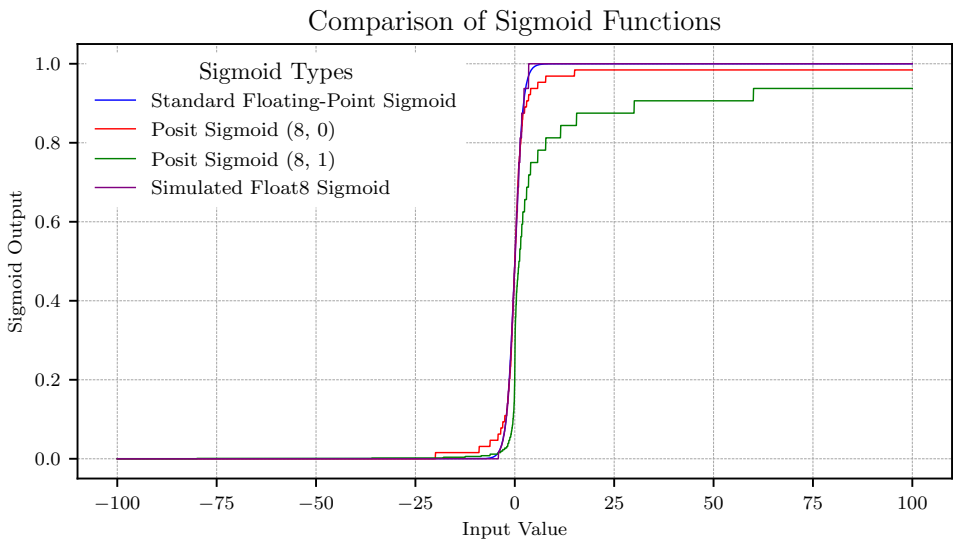
Sigmoid with 8-bit float types.

Efficient Sigmoid Activation

Sigmoid introduce crucial non-linearity [DQZ18]:

$$\sigma(x) = \frac{1}{1 + e^{-x}}$$

- *IEEE754* have aproximations tricks (0x5f3759df)
- *Posits* defend with **nearly 0 cost** sigmoid approximation
- MSB bitflip followed by constant bitshift



Sigmoid with 8-bit float types.

Current Progress

Posit AI acceleration

2.1 Executive Summary and Motivations

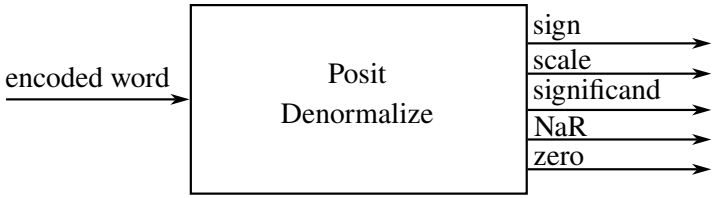
2.2 Posit Operator Framework (POF)

2.3 Machine Learning Evaluation

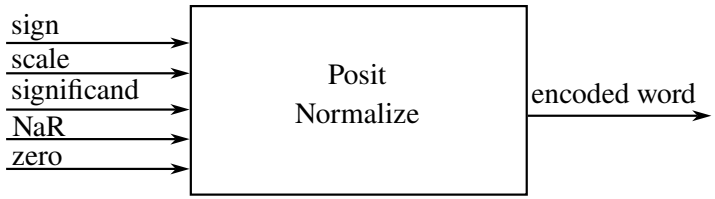
2.4 Conclusions

Bottom-Up Development: POF modules

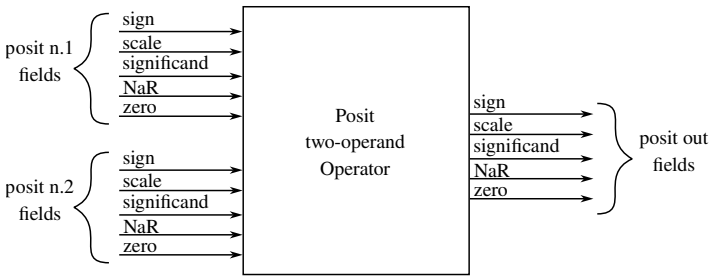
- Neurons and connections seen as a dependency graph.
- SDFG: Synchronous Data Flow Graph.
- **Streaming**
- Seamless chaining into **large IP**.
- Culminating in Fully Connected (FC) layer.
- **+8000** SystemVerilog lines of codes
- **50 Modules**



Posit Denormalize



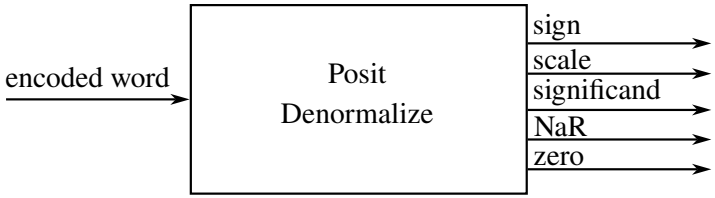
Posit Normalize



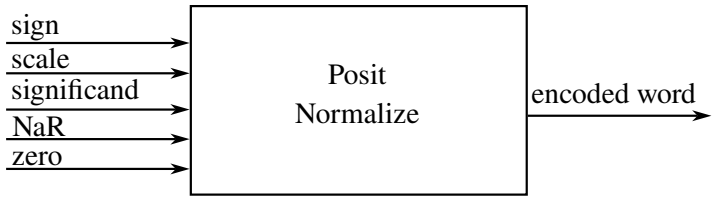
Posit Two Operands Operator

Bottom-Up Development: POF modules

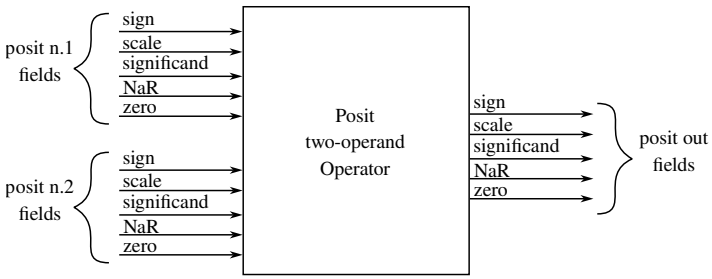
- Neurons and connections seen as a dependency graph.
- SDFG: Synchronous Data Flow Graph.
- **Streaming**
- Seamless chaining into **large IP**.
- Culminating in Fully Connected (FC) layer.
- **+8000** SystemVerilog lines of codes
- **50 Modules**



Posit Denormalize



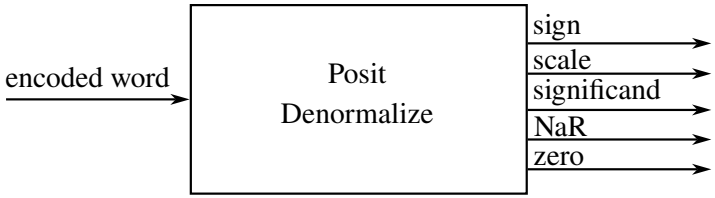
Posit Normalize



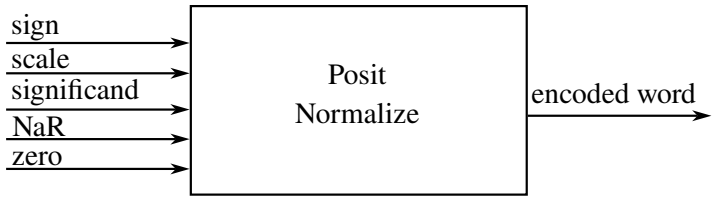
Posit Two Operands Operator

Bottom-Up Development: POF modules

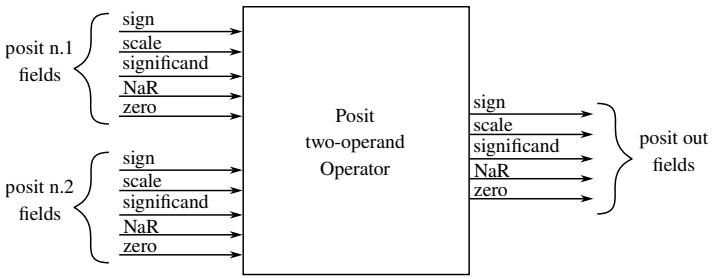
- Neurons and connections seen as a dependency graph.
- SDFG: Synchronous Data Flow Graph.
- **Streaming**
- Seamless chaining into **large IP**.
- Culminating in Fully Connected (FC) layer.
- **+8000** SystemVerilog lines of codes
- **50 Modules**



Posit Denormalize



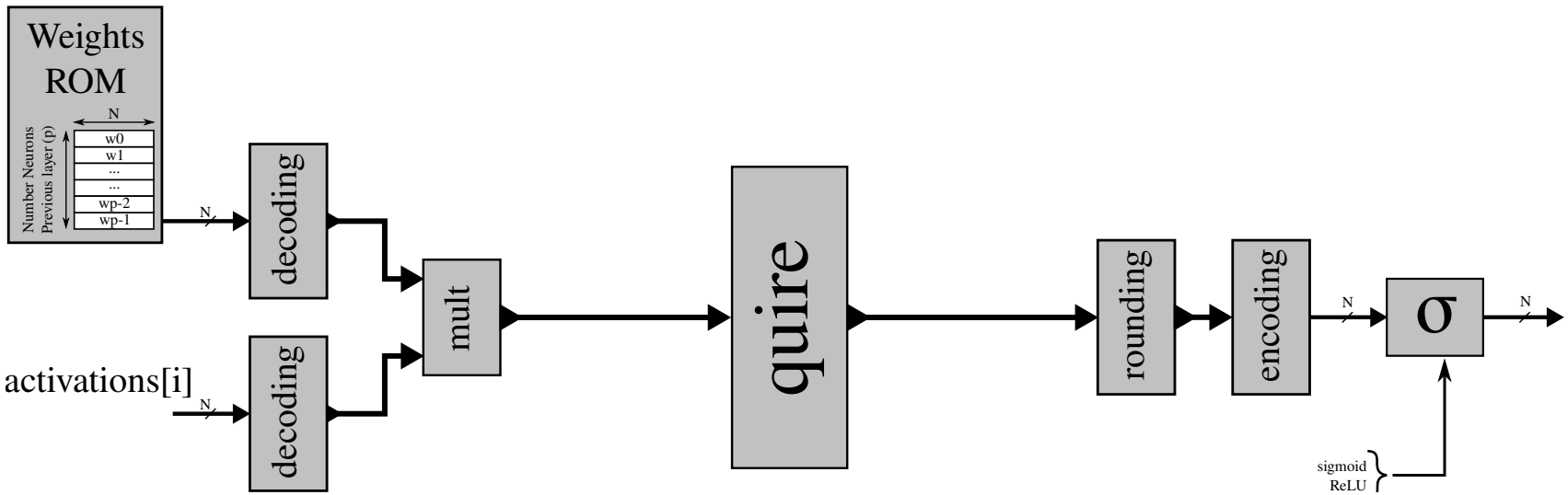
Posit Normalize



Posit Two Operands Operator

Neuron $\langle N, es \rangle$ Posit

- Weight-Activation dot-product, encoding/decoding, and nonlinear activations.



Quire / Exact

Hardware Cost Evaluation

Neurons Resource Utilization

Module	LUTs	FFs
Neuron<4,0,QUIRE>	116	72
Decoder<4,0>	3	0
Mult<4,0>	6	0
Quire19<4,0>	88	56
Encoder+Round<4,0>	7	0
Neuron<8,0,NOQUIRE>	336	126
Decoder<8,0>	27	0
Mult<8,0>	82	32
Adder<8,0>	133	57
Encoder+Round<8,0>	38	0
Neuron<8,0,QUIRE>	363	148
Decoder<8,0>	27	0
Mult<8,0>	85	51
Quire35<8,0>	213	97
Encoder+Round<8,0>	38	0

Utilization for Posit<16,0>.

Module	LUTs	FFs	DSP48
positron_layer	19482	5764	20
positron_inst (20x)	1002	279	1
pipeline_inst	125	52	0
posit_mult_inst	168	37	1
quire_prod_accum_inst	696	180	0
weights_ROM_inst	13	10	0

Utilization for Posit<32,0>.

Module name	LUTs	FFs	DSP48 Blocks
positron_layer	44321	11440	80
positron_inst (20x)	2220	546	4
pipeline_inst	277	100	0
posit_mult_inst	96	86	4
quire_prod_accum_inst	1833	347	0
weights_ROM_inst	14	10	0

Hardware Cost Evaluation

Neurons Resource Utilization

Module	LUTs	FFs
Neuron<4,0,QUIRE>	116	72
Decoder<4,0>	3	0
Mult<4,0>	6	0
Quire19<4,0>	88	56
Encoder+Round<4,0>	7	0
Neuron<8,0,NOQUIRE>	336	126
Decoder<8,0>	27	0
Mult<8,0>	82	32
Adder<8,0>	133	57
Encoder+Round<8,0>	38	0
Neuron<8,0,QUIRE>	363	148
Decoder<8,0>	27	0
Mult<8,0>	85	51
Quire35<8,0>	213	97
Encoder+Round<8,0>	38	0

Utilization for Posit<16,0>.

Module	LUTs	FFs	DSP48
positron_layer	19482	5764	20
positron_inst (20x)	1002	279	1
pipeline_inst	125	52	0
posit_mult_inst	168	37	1
quire_prod_accum_inst	696	180	0
weights_ROM_inst	13	10	0

Utilization for Posit<32,0>.

Module name	LUTs	FFs	DSP48 Blocks
positron_layer	44321	11440	80
positron_inst (20x)	2220	546	4
pipeline_inst	277	100	0
posit_mult_inst	96	86	4
quire_prod_accum_inst	1833	347	0
weights_ROM_inst	14	10	0

Hardware Cost Evaluation

Neurons Resource Utilization

Module	LUTs	FFs
Neuron<4,0,QUIRE>	116	72
Decoder<4,0>	3	0
Mult<4,0>	6	0
Quire19<4,0>	88	56
Encoder+Round<4,0>	7	0
Neuron<8,0,NOQUIRE>	336	126
Decoder<8,0>	27	0
Mult<8,0>	82	32
Adder<8,0>	133	57
Encoder+Round<8,0>	38	0
Neuron<8,0,QUIRE>	363	148
Decoder<8,0>	27	0
Mult<8,0>	85	51
Quire35<8,0>	213	97
Encoder+Round<8,0>	38	0

Utilization for Posit<16,0>.

Module	LUTs	FFs	DSP48
positron_layer	19482	5764	20
positron_inst (20x)	1002	279	1
pipeline_inst	125	52	0
posit_mult_inst	168	37	1
quire_prod_accum_inst	696	180	0
weights_ROM_inst	13	10	0

Utilization for Posit<32,0>.

Module name	LUTs	FFs	DSP48 Blocks
positron_layer	44321	11440	80
positron_inst (20x)	2220	546	4
pipeline_inst	277	100	0
posit_mult_inst	96	86	4
quire_prod_accum_inst	1833	347	0
weights_ROM_inst	14	10	0

Current Progress

Posit AI acceleration

2.1 Executive Summary and Motivations

2.2 Posit Operator Framework (POF)

2.3 Machine Learning Evaluation

2.4 Conclusions

Neural Network Topology

Topology Constraints

Three-layer Multi-Layer Perceptron (MLP) was dictated by initial hardware limitations.

Neural Network Topology

Topology Constraints

Three-layer Multi-Layer Perceptron (MLP) was dictated by initial hardware limitations.

- **Data set:** MNIST with 60,000/10,000 training/test images.
- **Input layer:** 784 neurons (28x28 pixels per image).
- **Hidden layer:** 20 neurons, sigmoid activation function $\sigma(x) = \frac{1}{1+e^{-x}}$.
- **Output layer:** 10 neurons for digit classification, sigmoid activation.

Neural Network Training and Evaluation

- **Learning rate:** Set at 0.2.
- **Cost function:** Cross-entropy.
- **Double-Precision Training**.
- **Top1 scores:** Training set: 88.59%; Test set: **91.73%**.

Neural Network Training and Evaluation

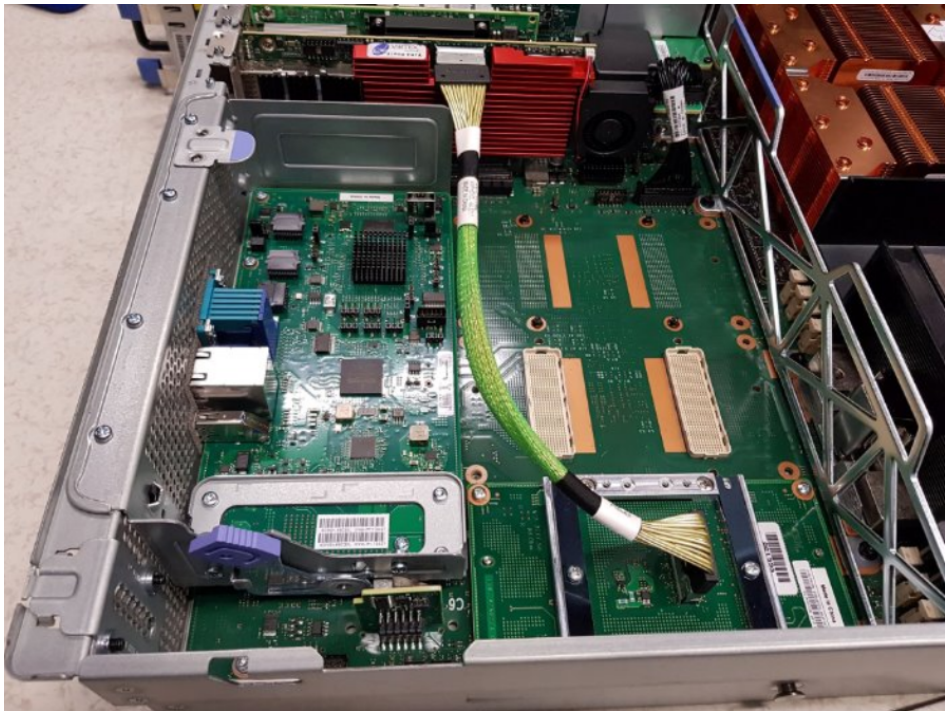
- **Learning rate:** Set at 0.2.
- **Cost function:** Cross-entropy.
- **Double-Precision Training**.
- **Top1 scores:** Training set: 88.59%; Test set: **91.73%**.

Top1 score Goal

The **91.73%** obtained by IEEE754-64 serves as an **oracle** and objective to **evaluate Posits**.

Hardware Platform Overview

- Power9 System (IBM AC922 with AlphaData 9V3 FPGA Board):
  - XCVU3P-2-FFVC1517 FPGA, UltraScale+ family.
  - AC922 POWER9 system, 40 cores (20 per socket), 2.3GHz.
- FPGA Specifications:
  - LUTs: 384k
  - Flip-Flops: 788k
  - DSP Slices: 2280
  - Block RAM: 25.3Mb
  - UltraRAM: 90Mb
- PCIe Gen4-8 lanes, **15.754 GB/s**, supports **CAPI2** and IBM SNAP = **512b@250MHz**.



IBM Power9 AC922; CAPI2 + VU3P.

Optimizing Throughput: Data Parallelism Approach

Challenge

One MLP consumes one datum per clock cycle. **Suboptimal** use of the high-speed link.

Our approach

Data-Parallelism by **interleaving** the pixels of input images. One MLP per independent images.

Example: Comparing standards

The **performance** of Posit VS. IEEE754 is **correlated** with the necessary **bitwidth** to reach the 91%

Optimizing Throughput: Data Parallelism Approach

Challenge

One MLP consumes one datum per clock cycle. **Suboptimal** use of the high-speed link.

Our approach

Data-Parallelism by **interleaving** the pixels of input images. One MLP per independent images.

Example: Comparing standards

The **performance** of Posit VS. IEEE754 is **correlated** with the necessary **bitwidth** to reach the 91%

Optimizing Throughput: Data Parallelism Approach

Challenge

One MLP consumes one datum per clock cycle. **Suboptimal** use of the high-speed link.

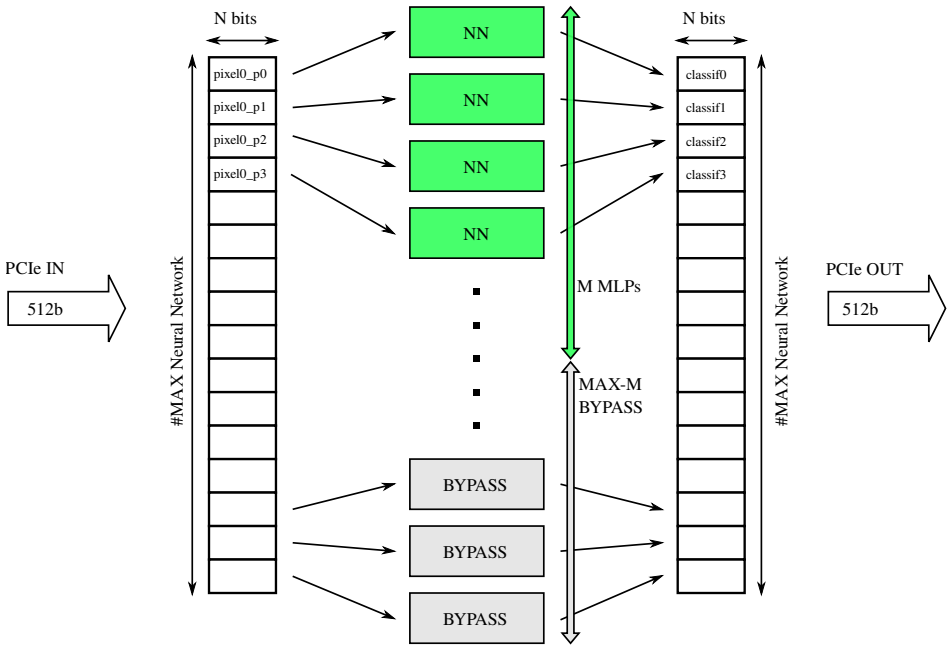
Our approach

Data-Parallelism by **interleaving** the pixels of input images. One MLP per independent images.

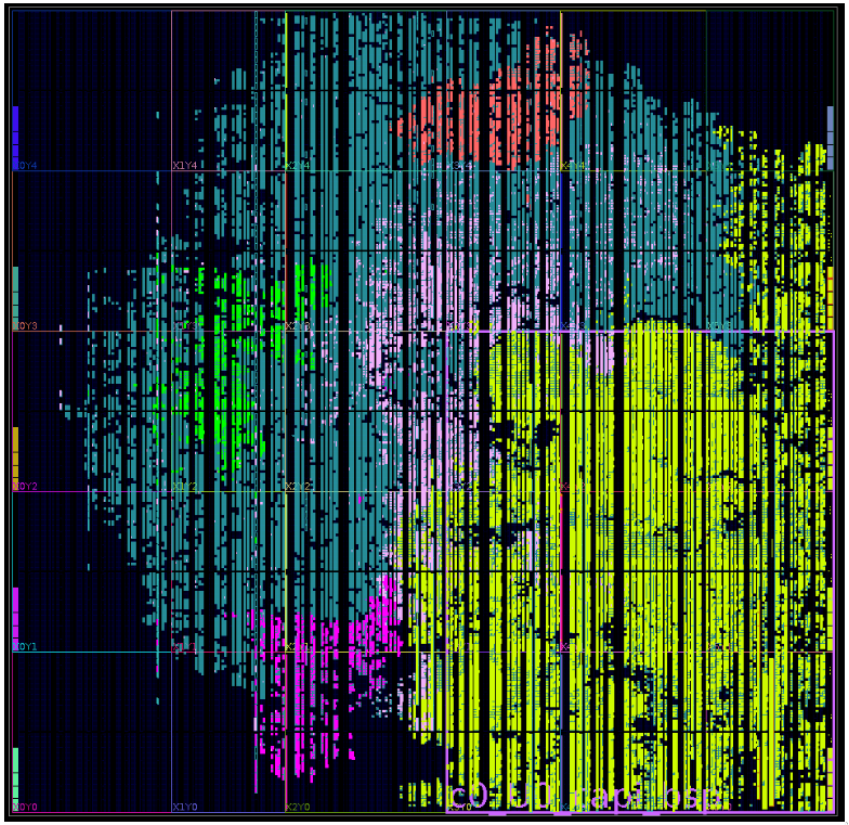
Example: Comparing standards

The **performance** of Posit VS. IEEE754 is **correlated** with the necessary **bitwidth** to reach the 91%

Data-Level Parallelism Examples



Data parallelism via interleaving pixels across MLP instances.



16 MLPs of posit<8,0>, utilizing 25% of bandwidth (128/512).

Images per second VS. Accuracy VS. Footprints

16 MLPs: Saving Communication

Format	Accuracy	Memory	Throughput	Bandwidth	FPS
posit<4,0,quire>	91%	23.5MB	1.56GB/s	12.5%	3.8 $\times 10^6$
posit<8,0,quire>	91%	47MB	3.1GB/s	25%	3.8 $\times 10^6$
posit<16,0,quire>	91%	94MB	6.2GB/s	50%	3.8 $\times 10^6$
IEEE<4,1>	10%	23.5MB	1.56GB/s	12.5%	3.8 $\times 10^6$
IEEE<8,4>	50%	47MB	3.1GB/s	25%	3.8 $\times 10^6$
IEEE<16,5>	91%	94MB	6.2GB/s	50%	3.8 $\times 10^6$
IEEE<32,8>	91%	189MB	12.5GB/s	100%	3.8 $\times 10^6$

Theoretical Performance in an ideal FPGA

Format	Accuracy (Top1 Score)	Memory Size	# of MLPs	Bandwidth Occupancy	FPS
posit<4,0,quire>	91%	189MB	128	100%	30.6$\times 10^6$
posit<8,0,quire>	91%	189MB	64	100%	15.3 $\times 10^6$
posit<16,0,quire>	91%	189MB	32	100%	7.65 $\times 10^6$
IEEE<4,1>	10%	189MB	128	100%	30.6 $\times 10^6$
IEEE<8,4>	50%	189MB	64	100%	15.3 $\times 10^6$
IEEE<16,5>	91%	189MB	32	100%	7.65 $\times 10^6$
IEEE<32,8>	91%	189MB	16	100%	3.8 $\times 10^6$

Current Progress

Posit AI acceleration

2.1 Executive Summary and Motivations

2.2 Posit Operator Framework (POF)

2.3 Machine Learning Evaluation

2.4 Conclusions

Summary of Insights and Key Contributions

POWER9 + CAPI2

Successful **SW/HW co-design** in a cutting-edge environment, maximizing bandwidth utilization.

Posit Development

Provision of an **open-source framework** with a comprehensive suite of Posit operators for **building neural networks**.

Best Performance

Notably, Posit<4,0,QUIRE> is $4\times$ **more efficient** and uses $4\times$ **less memory** than the best IEEE754 float.

Findings

Small Emerging format as transport combined with **extended internal precision** is promising. **Alleviates the walls (A)**.

Summary of Insights and Key Contributions

POWER9 + CAPI2

Successful **SW/HW co-design** in a cutting-edge environment, maximizing bandwidth utilization.

Posit Development

Provision of an **open-source framework** with a comprehensive suite of Posit operators for **building neural networks**.

Best Performance

Notably, Posit<4,0,QUIRE> is $4\times$ **more efficient** and uses $4\times$ **less memory** than the best IEEE754 float.

Findings

Small Emerging format as transport combined with **extended internal precision** is promising. **Alleviates the walls (A)**.

Discussions

Manual VS. Automated Pipelining

Transitioning to **automated pipeline tools**, especially when adapting to multiple computer formats.

Expanding Model Complexity

Integrating more impactful models like **CNNs** and generic HPC kernels such as **GEMM**, to enhance the scope and **impact of our research**.

Fair Format Comparison

Enabling **fair comparisons** between different computational formats by ensuring feature parity and maintaining **format agnosticism**.

Current Progress

“A Generator of Numerically-Tailored Accelerators for Batched GEMMs”

Generator of Systolic Arrays

3.1 Executive Summary

3.2 Systolic Array Generator

3.3 Evaluation

3.4 Conclusions

Executive Summary

General Matrix Multiplication kernel acceleration

A **large variety of scientific applications** rely on **GEMM** multiplication kernel. The numerical requirements are **very heterogeneous**, e.g., AI or weather forecasting.

State-of-the-Art Accelerators

Matrix-Matrix Multiplication (MMM) accelerators **have no focus on precision/accuracy**, but on performance.

Our Proposal

Generation of scalable Systolic Arrays with **adaptative precision** Processing Elements.

Executive Summary

General Matrix Multiplication kernel acceleration

A **large variety of scientific applications** rely on **GEMM** multiplication kernel. The numerical requirements are **very heterogeneous**, e.g., AI or weather forecasting.

State-of-the-Art Accelerators

Matrix-Matrix Multiplication (MMM) accelerators **have no focus on precision/accuracy**, but on performance.

Our Proposal

Generation of scalable Systolic Arrays with **adaptative precision** Processing Elements.

Executive Summary

General Matrix Multiplication kernel acceleration

A **large variety of scientific applications** rely on **GEMM** multiplication kernel. The numerical requirements are **very heterogeneous**, e.g., AI or weather forecasting.

State-of-the-Art Accelerators

Matrix-Matrix Multiplication (MMM) accelerators **have no focus on precision/accuracy**, but on performance.

Our Proposal

Generation of scalable Systolic Arrays with **adaptative precision** Processing Elements.

Current Progress

Generator of Systolic Arrays

3.1 Executive Summary

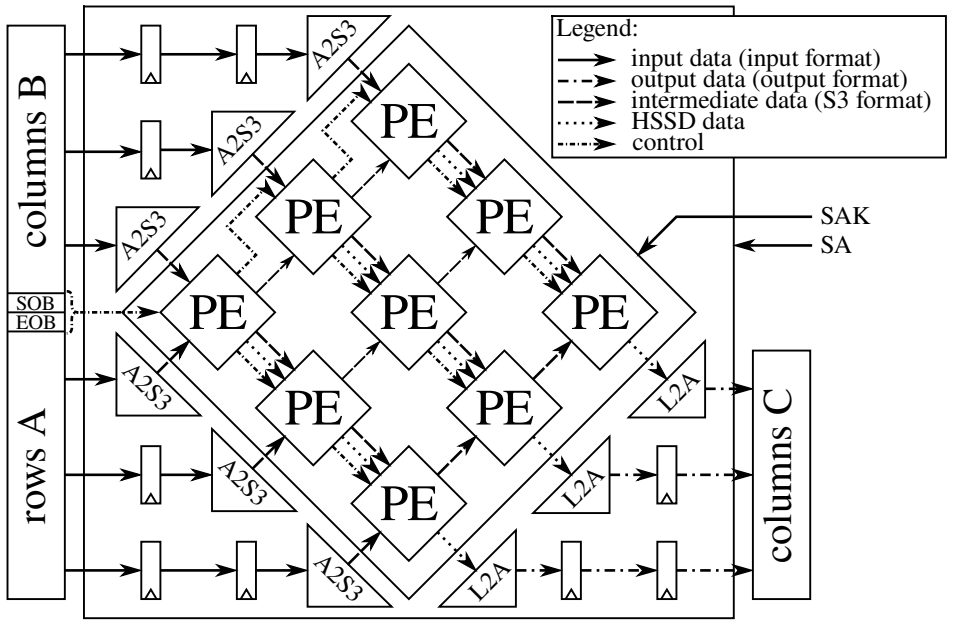
3.2 Systolic Array Generator

3.3 Evaluation

3.4 Conclusions

Systolic Array Generator: Overview

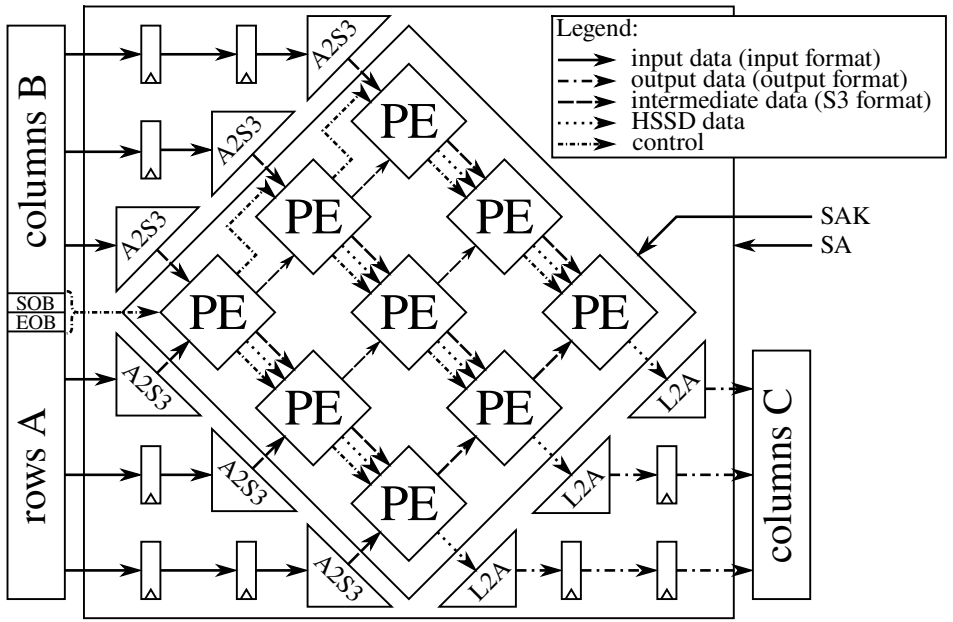
- BLAS level 3 GEMM routine
- Generator 2D Systolic Arrays
- Target agnostic
- Performance:
  - Scalable
  - HSSD Extraction scheme
- **Arithmetic focus**:
  - S3: Computer format agnostic
  - 3 Accumulators



3×3 Systolic Array.

Systolic Array Generator: Overview

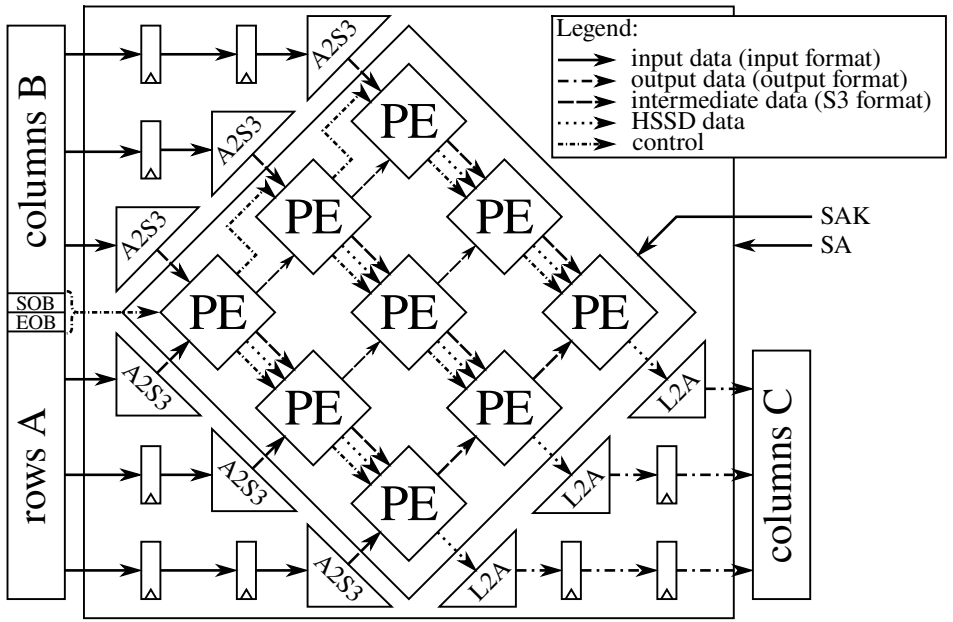
- BLAS level 3 GEMM routine
- Generator 2D Systolic Arrays
- Target agnostic
- Performance:
  - Scalable
  - HSSD Extraction scheme
- **Arithmetic focus**:
  - S3: Computer format agnostic
  - 3 Accumulators



3×3 Systolic Array.

Systolic Array Generator: Overview

- BLAS level 3 GEMM routine
- Generator 2D Systolic Arrays
- Target agnostic
- Performance:
  - Scalable
  - HSSD Extraction scheme
- **Arithmetic focus**:
  - S3: Computer format agnostic
  - 3 Accumulators



3×3 Systolic Array.

Systolic Array Generator: S3 Format

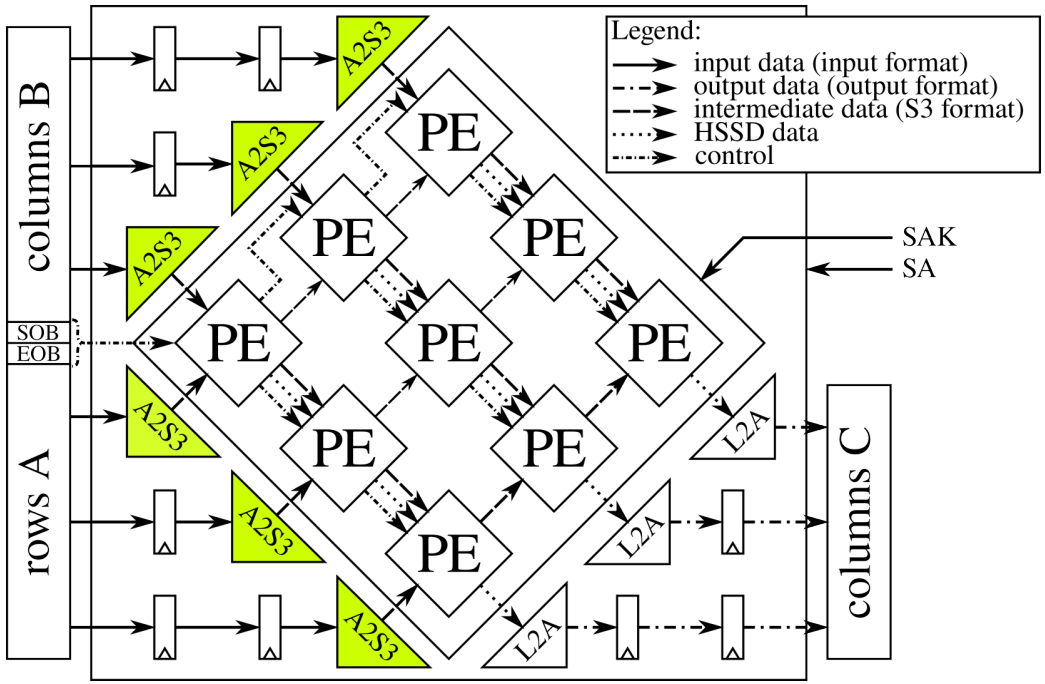
- **S**ign **S**cale **S**ignificand
- All our datapaths are in **S3** format
- **Arithmetic Agnostic** once translated/mapped
- Simple arithmetic:
  - necessary and sufficient bits
  - Eliminate redundancy, dual zeros, special cases
- A quintuple format: ⟨NaN, Sign, Scale, Implicit, Fraction⟩

Common computer number formats and their S3 translations.

Comp. Arith.	Value	$S3\ \omega S$	Bias	$S3\ \omega F$	S3 Quintuple
IEEE-754 16	0	5	$2^{\omega S-1}-1=15$	10	$\langle 0, x, xxxxx, 0, 0000000000\rangle$
Posit $\langle 8,0\rangle$	1	4	$(N-2)\cdot 2^{es}=6$	5	$\langle 0, 0, 0110, 1, 00000\rangle$
Posit $\langle 16,2\rangle$	NaN	7	$(N-2)\cdot 2^{es}=56$	11	$\langle 1, x, xxxxxxx, x, xxxxxxxxxxx\rangle$
bfloat16	3.5	8	$2^{\omega S-1}-1=127$	7	$\langle 0, 0, 10000000, 1, 1100000\rangle$

Systolic Array: Building Blocks

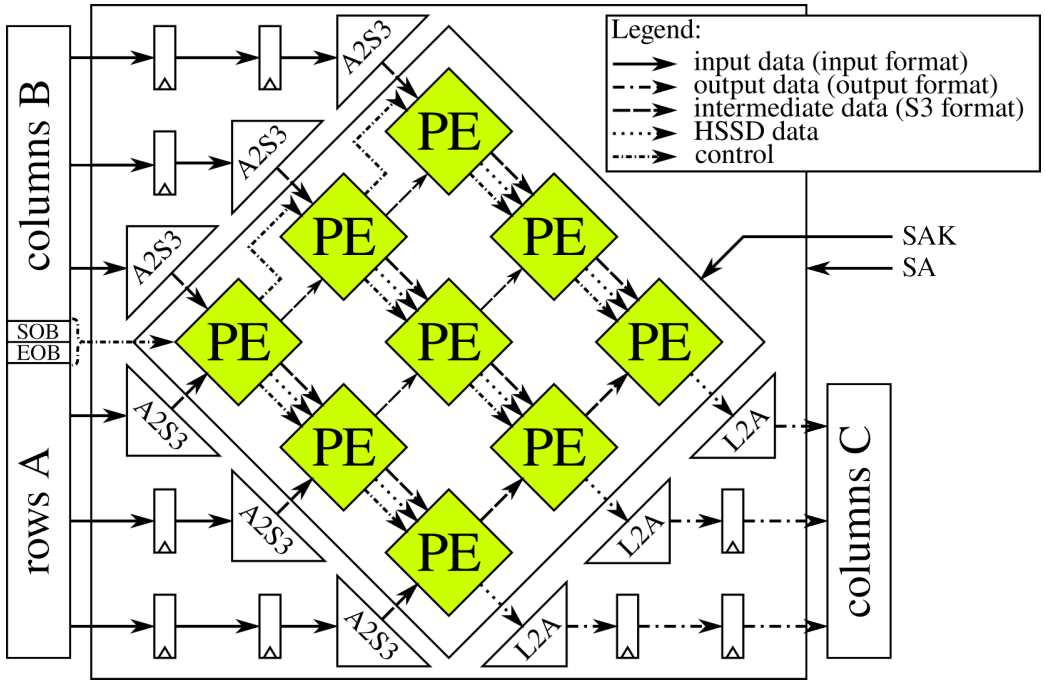
- **A2S3**: Arithmetic to S3 - Translates any arithmetic format into **S3**
- **PEs**: Processing Elements - Include the **S3** Fused Dot Product (**S3FDP**)
- **L2A**: Large Accumulator to Arithmetic - output results in the chosen format (**exact**, **same**, **specific**).



Systolic Array key components.
$n + m$ "A2S3"

Systolic Array: Building Blocks

- **A2S3**: Arithmetic to S3 - Translates any arithmetic format into **S3**
- **PEs**: Processing Elements - Include the **S3** Fused Dot Product (**S3FDP**)
- **L2A**: Large Accumulator to Arithmetic - output results in the chosen format (**exact**, **same**, **specific**).

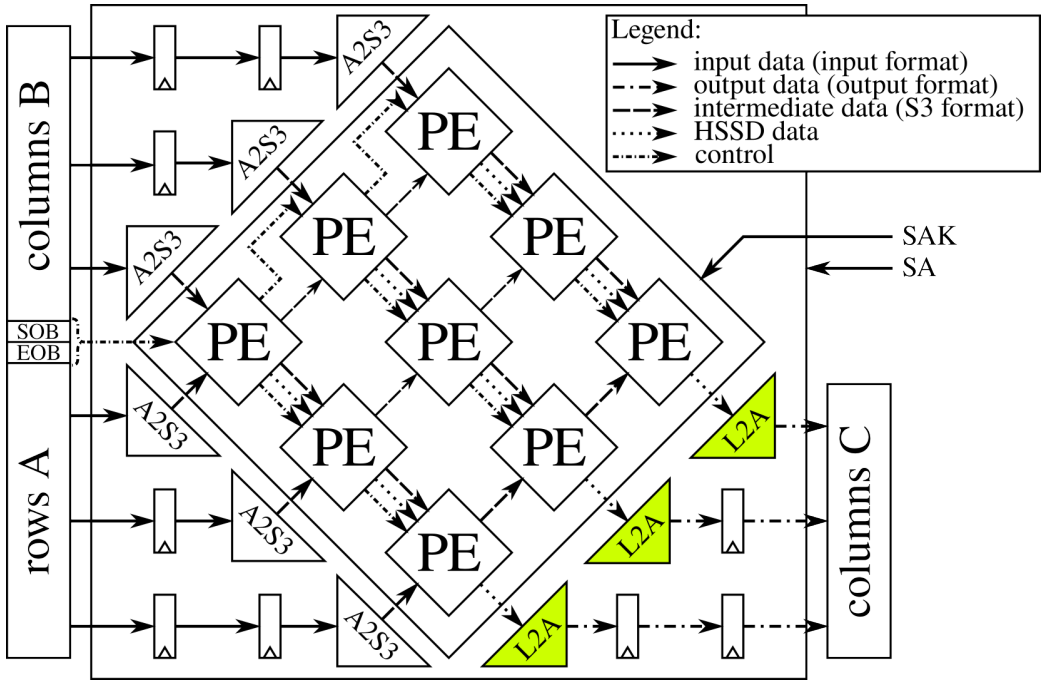


Systolic Array key components.

$n * m$ "PE"

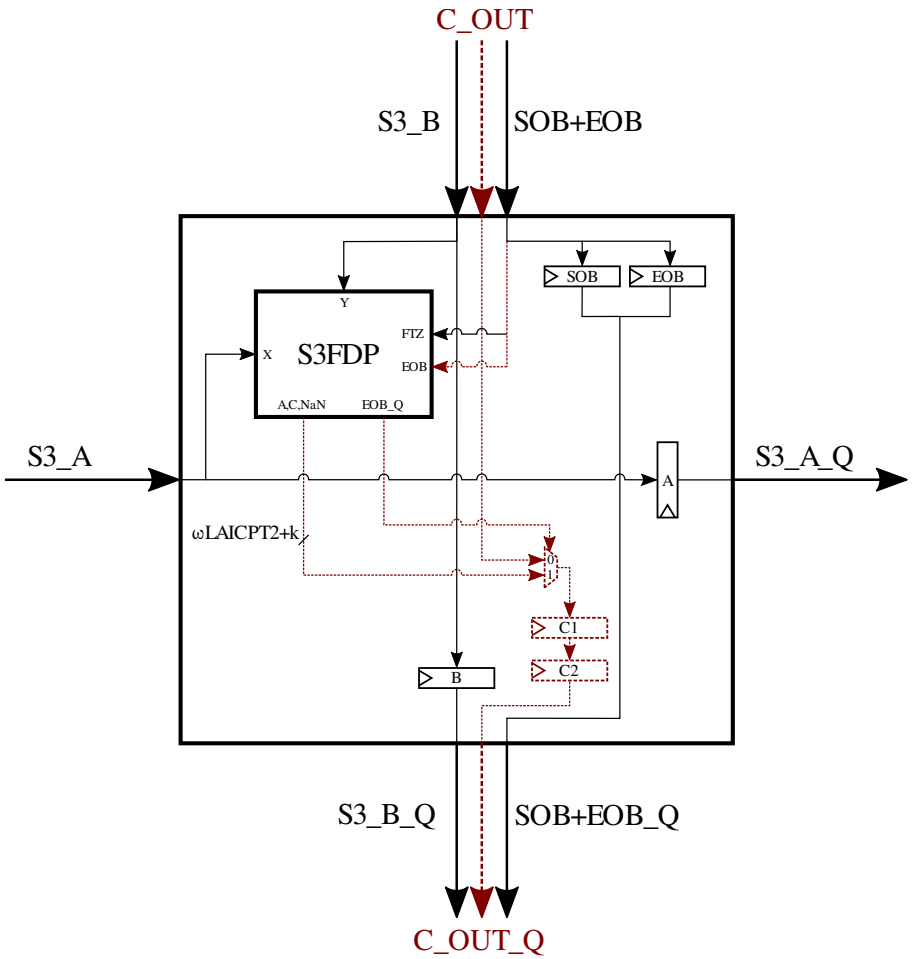
Systolic Array: Building Blocks

- **A2S3**: Arithmetic to S3 - Translates any arithmetic format into **S3**
- **PEs**: Processing Elements - Include the **S3** Fused Dot Product (**S3FDP**)
- **L2A**: Large Accumulator to Arithmetic - output results in the chosen format (**exact**, **same**, **specific**).



Systolic Array key components.
m "L2A"

Processing Element (PE)



Schematic of a PE. HSSD highlighted.

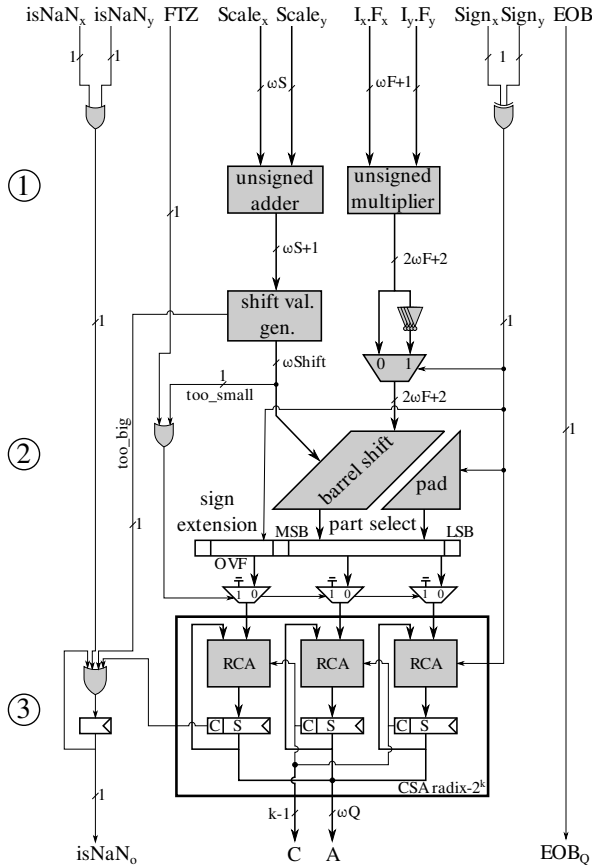
Algorithm 5 Local program view of a PE.

```
1:  module PE: (instantiates S3FDP s)
2:      S3_A_Q ← S3_A
3:      S3_B_Q ← S3_B                          } 1.Passing data
4:      SOB_Q ← SOB
5:      EOB_Q ← EOB                            } 2.Distributed control
6:      s.(x) ⇐ S3_A
7:      s.(y) ⇐ S3_B
8:      s.(FTZ) ⇐ SOB                          } 3.Fused Dot Product
9:      s.(EOB) ⇐ EOB
10:     C_OUT_Q ← C2
11:     C2 ← C1
12:     if s.(EOB_Q == 1'b1) then
13:         C1 ← s.({A, NaN, C})               } 4.HSSD
14:     else
15:         C1 ← C_OUT
16:     end if
17: endmodule
```

S3 Fused-Dot-Product (S3FDP)

Adaptative Kulisch Fixed-Point Accumulator [DPC+08]

1. Exact Product into extended S3
2. S3 translation and alignment to fixed-point
3. Accumulation with RCA or CSA (radix-2^k approach)

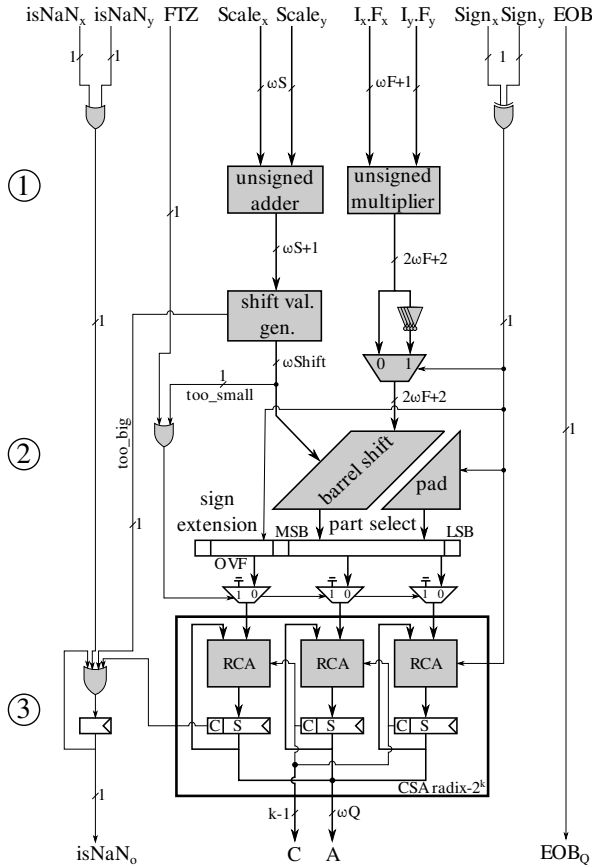


Circuit design of a generic S3FDP

S3 Fused-Dot-Product (S3FDP)

Adaptative Kulisch Fixed-Point Accumulator [DPC+08]

1. Exact Product into extended S3
2. S3 translation and alignment to fixed-point
3. Accumulation with RCA or CSA (radix-2^k approach)

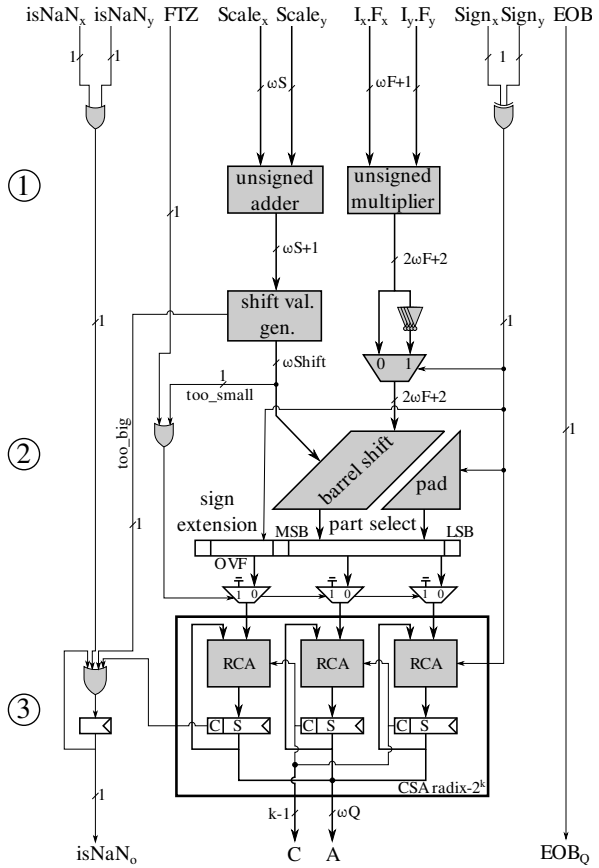


Circuit design of a generic S3FDP

S3 Fused-Dot-Product (S3FDP)

Adaptative Kulisch Fixed-Point Accumulator [DPC+08]

1. Exact Product into extended S3
2. S3 translation and alignment to fixed-point
3. Accumulation with RCA or CSA (radix-2^k approach)



Circuit design of a generic S3FDP

Current Progress

Generator of Systolic Arrays

3.1 Executive Summary

3.2 Systolic Array Generator

3.3 Evaluation

3.4 Conclusions

Design Space Exploration (DSE)

22 arithmetic formats

bfloat16, "ieee8", ieee16, ieee32, ieee64, tfp8, tfp16, tfp32, tfp64, posit<4,0>, posit<8,0>, posit<8,1>, posit<8,2>, posit<16,0>, posit<16,1>, posit<16,2>, posit<32,0>, posit<32,1>, posit<32,2>, posit<64,1>, posit<64,2>, posit<64,3>.

3 Accumulator Configurations

α - Small accumulator with small MSB for AI; β - Exact accumulator (Kulisch and Quire), preventing roundoff errors; γ - 100-bit accumulator.

Combinations

A total of 66 combinations (arithmetic, accumulator) explored in our design space exploration.

Design Space Exploration (DSE)

22 arithmetic formats

bfloat16, "ieee8", ieee16, ieee32, ieee64, tfp8, tfp16, tfp32, tfp64, posit<4,0>, posit<8,0>, posit<8,1>, posit<8,2>, posit<16,0>, posit<16,1>, posit<16,2>, posit<32,0>, posit<32,1>, posit<32,2>, posit<64,1>, posit<64,2>, posit<64,3>.

3 Accumulator Configurations

α - Small accumulator with small MSB for AI; β - Exact accumulator (Kulisch and Quire), preventing roundoff errors; γ - 100-bit accumulator.

Combinations

A total of 66 combinations (arithmetic, accumulator) explored in our design space exploration.

Design Space Exploration (DSE)

22 arithmetic formats

bfloat16, "ieee8", ieee16, ieee32, ieee64, tfp8, tfp16, tfp32, tfp64, posit<4,0>, posit<8,0>, posit<8,1>, posit<8,2>, posit<16,0>, posit<16,1>, posit<16,2>, posit<32,0>, posit<32,1>, posit<32,2>, posit<64,1>, posit<64,2>, posit<64,3>.

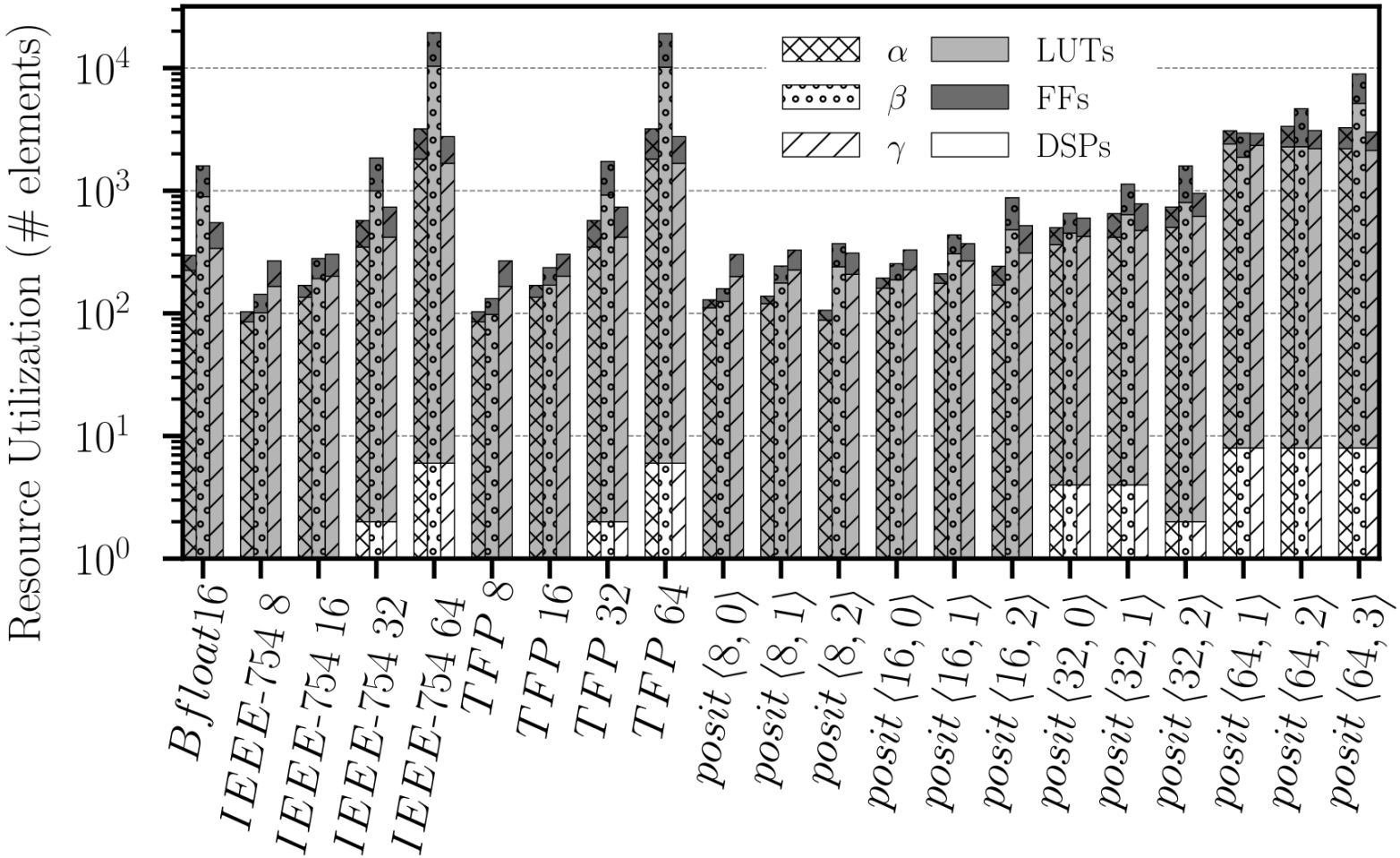
3 Accumulator Configurations

α - Small accumulator with small MSB for AI; β - Exact accumulator (Kulisch and Quire), preventing roundoff errors; γ - 100-bit accumulator.

Combinations

A total of 66 combinations (arithmetic, accumulator) explored in our design space exploration.

Design Space Exploration Results



Routed resource of S3FDPs@250MHz

IEEE754 comparison

Hardware cost of "IEEE754" S3 vs. Xilinx FMA IP.

8-bit configurations

Config.	α	β	γ	*FMA*
LUTs	85	109	166	233
FFs	18	42	102	342
DSPs	0	0	0	1
CARRY8s	2	6	13	17
Power(PE)(W)	0.003	0.004	0.006	0.016
Power(system)(W)	2.828	4.807	9.860	15.872
En. Eff.(system)(GFlops/s/W)	175.4	103.2	50.3	31.3

16-bit configurations

Config.	α	β	γ	*FMA*
LUTs	134	187	200	425
FFs	34	124	102	680
DSPs	1	1	1	1
CARRY8s	4	16	13	22
Power(PE)(W)	0.008	0.018	0.014	0.032
Power(system)(W)	2.121	3.177	3.749	7.680
En. Eff.(system)(GFlops/s/W)	56.6	37.8	32.0	15.6

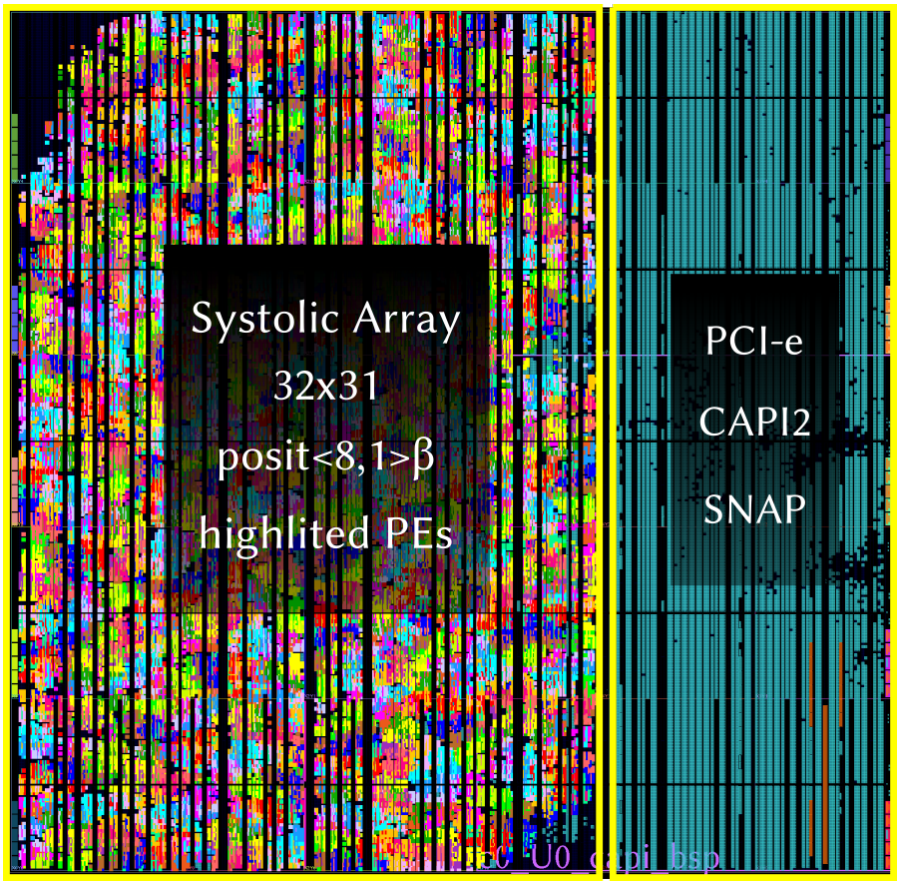
32-bit configurations

Config.	α	β	γ	*FMA*
LUTs	345	1039	416	769
FFs	225	865	317	1278
DSPs	2	2	2	2
CARRY8s	12	19	37	73
Power(PE)(W)	0.019	0.058	0.023	0.066
Power(system)(W)	1.233	5.329	1.554	3.696
En. Eff.(system)(GFlops/s/W)	22.7	5.6	18.09	7.6

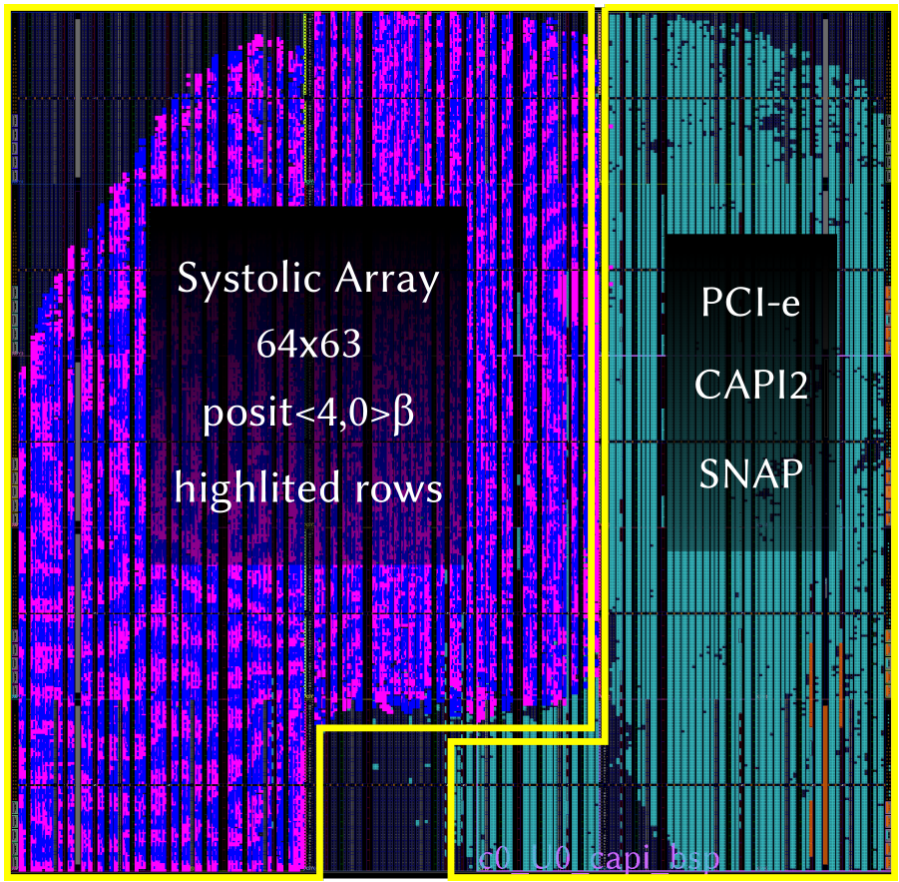
64-bit configurations

Config.	α	β	γ	*FMA*
LUTs	1806	11414	1668	1785
FFs	1383	8996	1089	2884
DSPs	6	6	6	10
CARRY8s	38	272	34	70
Power(PE)(W)	0.085	0.460	0.081	0.191
Power(system)(W)	1.002	13.633	0.971	2.292
En. Eff.(system)(GFlops/s/W)	6.0	0.4	6.2	2.6

DSE: Scalability of PnR designs



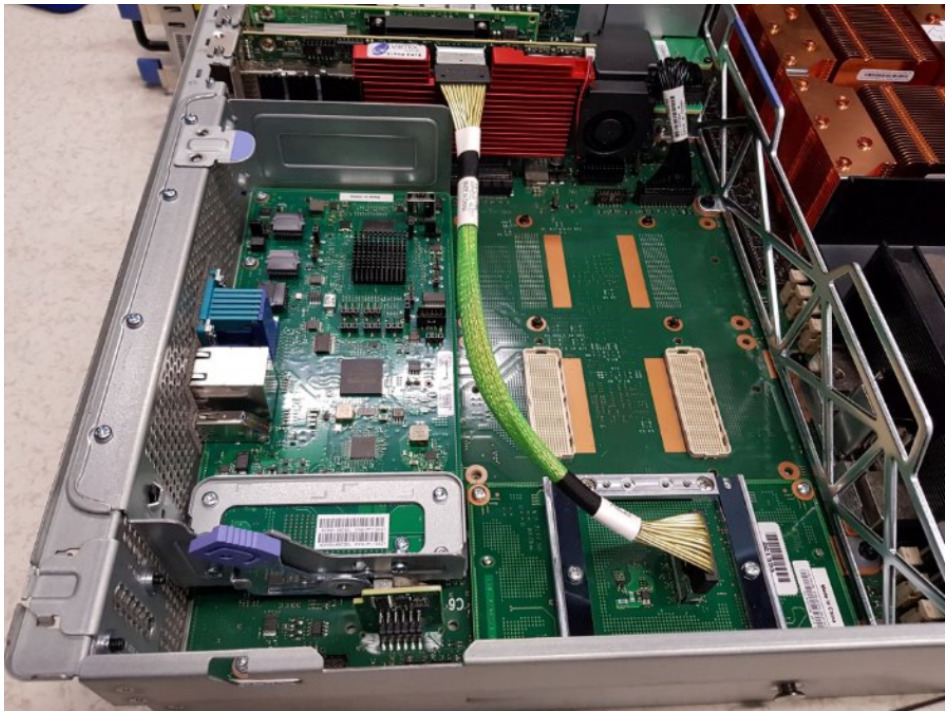
32×31 $posit\langle 8, 1\rangle\beta$ array



64×63 $posit\langle 4, 0\rangle\beta$ array

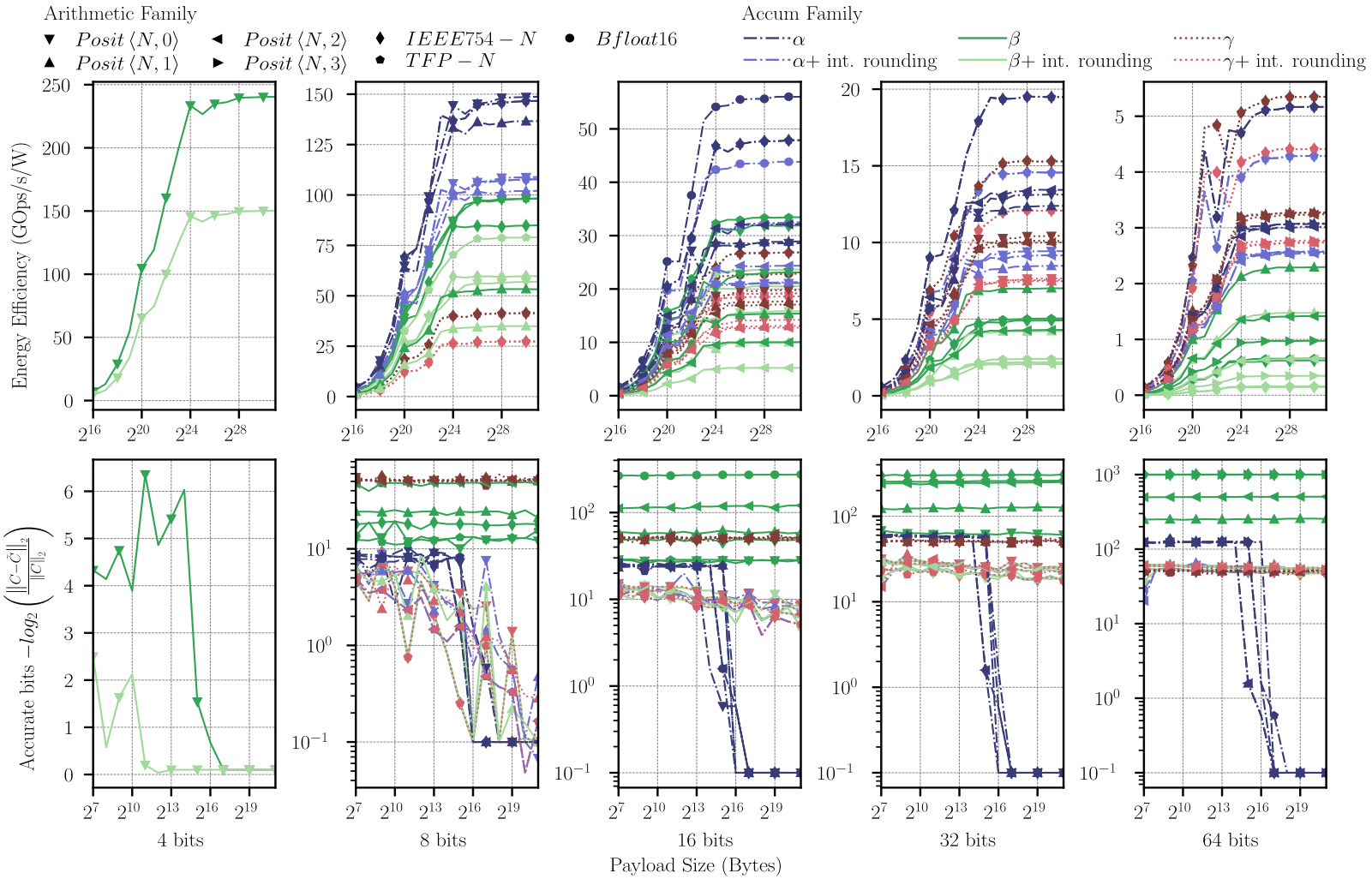
Hardware Platform Overview

- Power9 System (IBM AC922 with AlphaData 9V3 FPGA Board):
  - XCVU3P-2-FFVC1517 FPGA, UltraScale+ family.
  - AC922 POWER9 system, 40 cores (20 per socket), 2.3GHz.
- FPGA Specifications:
  - LUTs: 384k
  - Flip-Flops: 788k
  - DSP Slices: 2280
  - Block RAM: 25.3Mb
  - UltraRAM: 90Mb
- PCIe Gen4-8 lanes, 15.754 GB/s, supports CAPI2 and IBM SNAP.



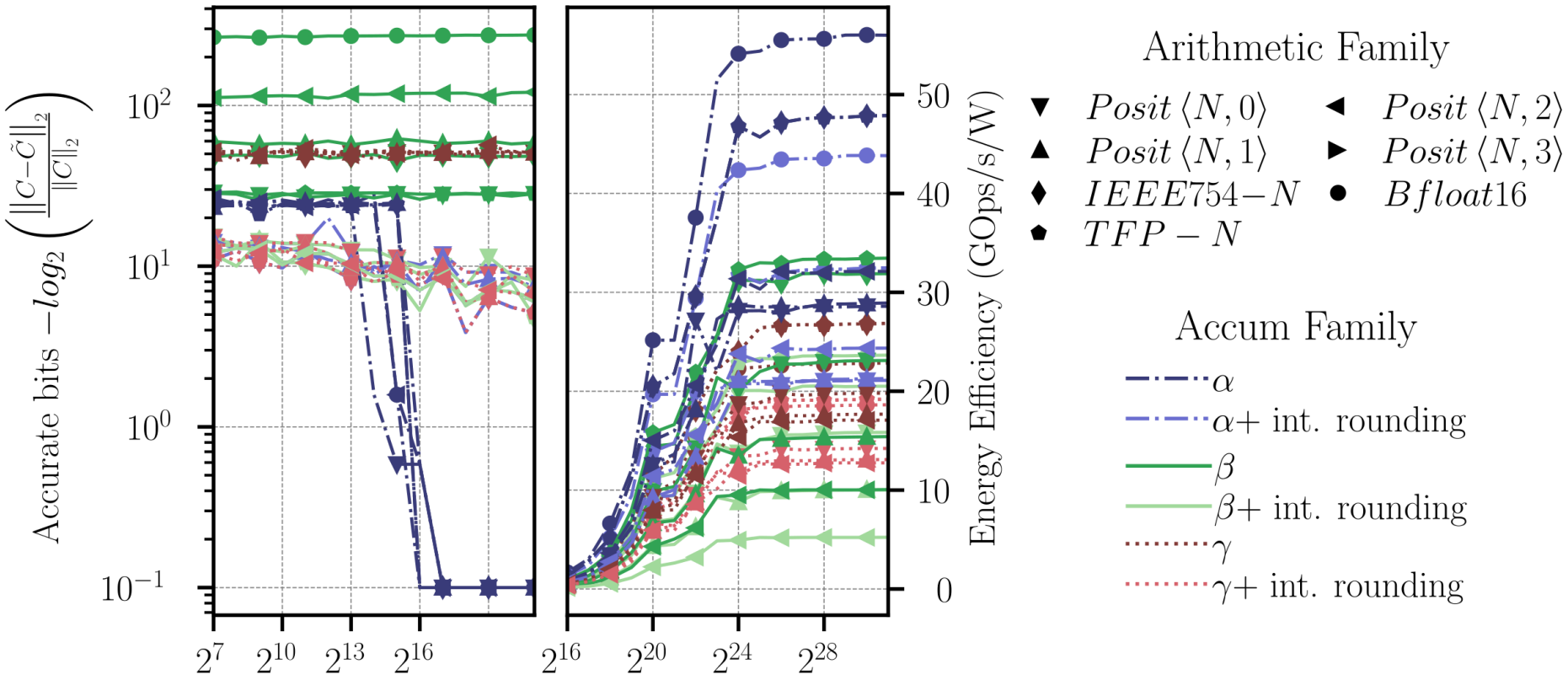
IBM Power9 AC922 ; CAPI2 + VU3P.

Evaluation: Energy Efficiency VS Accuracy



Energy efficiency (top) against Accuracy (bottom)

Zoom on 16-bit: Energy Efficiency VS Accuracy



Energy eff. and Accuracy for different payload sizes.

Current Progress

Generator of Systolic Arrays

Summary of Insights and Key Contributions

Large Design Space Exploration

Conducting extensive design space exploration and evaluation leading to **several profiles of GEMM kernels**.

State-of-the-art Enhancement

Improving single-precision energy efficiency of FPGA GEMM accelerators by $\mathbf{1.86}\times$.

Answering Sparsity of numerical Requirements

Our accelerators achieve **240 GOps/s/W with 4 accurate** bits to **0.65 GOps/s/W with over 1000** accurate bits, serving a **broad range of applications**.

Findings

Internal **precision budgeting** against a workload **alleviates "the walls"**.

Summary of Insights and Key Contributions

Large Design Space Exploration

Conducting extensive design space exploration and evaluation leading to **several profiles of GEMM kernels**.

State-of-the-art Enhancement

Improving single-precision energy efficiency of FPGA GEMM accelerators by **1.86**×.

Answering Sparsity of numerical Requirements

Our accelerators achieve **240 GOps/s/W with 4 accurate** bits to **0.65 GOps/s/W with over 1000** accurate bits, serving a **broad range of applications**.

Findings

Internal **precision budgeting** against a workload **alleviates "the walls"**.

Summary of Insights and Key Contributions

Large Design Space Exploration

Conducting extensive design space exploration and evaluation leading to **several profiles of GEMM kernels**.

State-of-the-art Enhancement

Improving single-precision energy efficiency of FPGA GEMM accelerators by **1.86**×.

Answering Sparsity of numerical Requirements

Our accelerators achieve **240 GOps/s/W with 4 accurate** bits to **0.65 GOps/s/W with over 1000** accurate bits, serving a **broad range of applications**.

Findings

Internal **precision budgeting** against a workload **alleviates "the walls"**.

Summary of Insights and Key Contributions

Large Design Space Exploration

Conducting extensive design space exploration and evaluation leading to **several profiles of GEMM kernels**.

State-of-the-art Enhancement

Improving single-precision energy efficiency of FPGA GEMM accelerators by **1.86**×.

Answering Sparsity of numerical Requirements

Our accelerators achieve **240 GOps/s/W with 4 accurate** bits to **0.65 GOps/s/W with over 1000** accurate bits, serving a **broad range of applications**.

Findings

Internal **precision budgeting** against a workload **alleviates "the walls"**.

Discussions

Workload Discussion

The range $[-1, 1]$ does not reflect real workload conditions. Other contributions propose evaluations with **artificial** matrices in ranges like $[-10, 10]$, $[-100, 100]$, up to $[-10^8, 10^8]$.

Future Research Directions

Adopt a more **realistic** approach by bridging to software-based HPC code.

Discussions

Workload Discussion

The range $[-1, 1]$ does not reflect real workload conditions. Other contributions propose evaluations with **artificial** matrices in ranges like $[-10, 10]$, $[-100, 100]$, up to $[-10^8, 10^8]$.

Future Research Directions

Adopt a more **realistic** approach by bridging to software-based HPC code.

Current Progress

"An Open-Source Framework for Efficient Numerically-Tailored Computations"

Numerically-tailored BLAS

Executive Summary

GEMM Multiplication Kernel

A large variety of scientific applications rely on **GEMM** kernel. The **numerical requirements are very heterogeneous**, e.g., AI or weather forecasting.

Lack of Hardware Precision Adaptability

Extended or adaptative **software precision is expensive** and often the computations land in commodity hardware.

Our Proposal

We propose an **Open-Source framework** to bridge high-level libraries to numerically aware and accelerated GEMM kernels.

Executive Summary

GEMM Multiplication Kernel

A large variety of scientific applications rely on **GEMM** kernel. The **numerical requirements are very heterogeneous**, e.g., AI or weather forecasting.

Lack of Hardware Precision Adaptability

Extended or adaptative **software precision is expensive** and often the computations land in commodity hardware.

Our Proposal

We propose an **Open-Source framework** to bridge high-level libraries to numerically aware and accelerated GEMM kernels.

Executive Summary

GEMM Multiplication Kernel

A large variety of scientific applications rely on **GEMM** kernel. The **numerical requirements are very heterogeneous**, e.g., AI or weather forecasting.

Lack of Hardware Precision Adaptability

Extended or adaptative **software precision is expensive** and often the computations land in commodity hardware.

Our Proposal

We propose an **Open-Source framework** to bridge high-level libraries to numerically aware and accelerated GEMM kernels.

Current Progress

Numerically-tailored BLAS

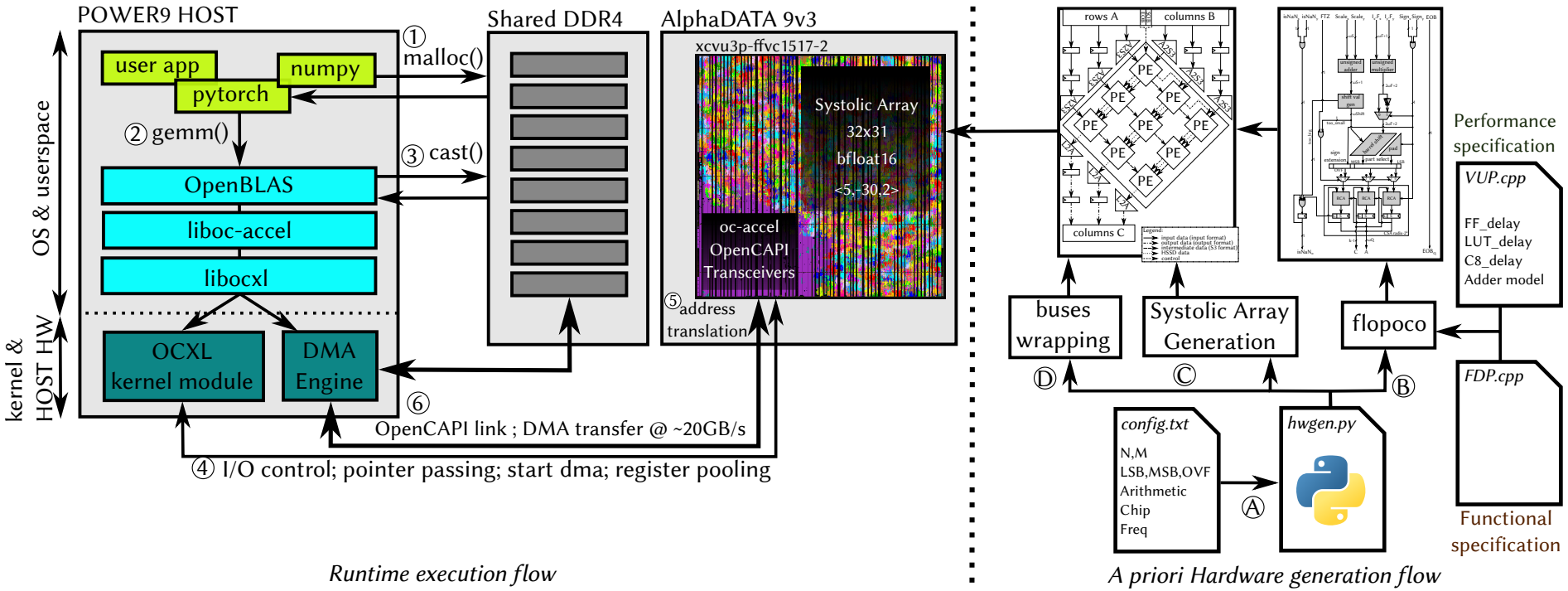
4.1 Executive Summary

4.2 Open Source SW/HW Co-designed Framework

4.3 Workload Evaluation

4.4 Conclusions

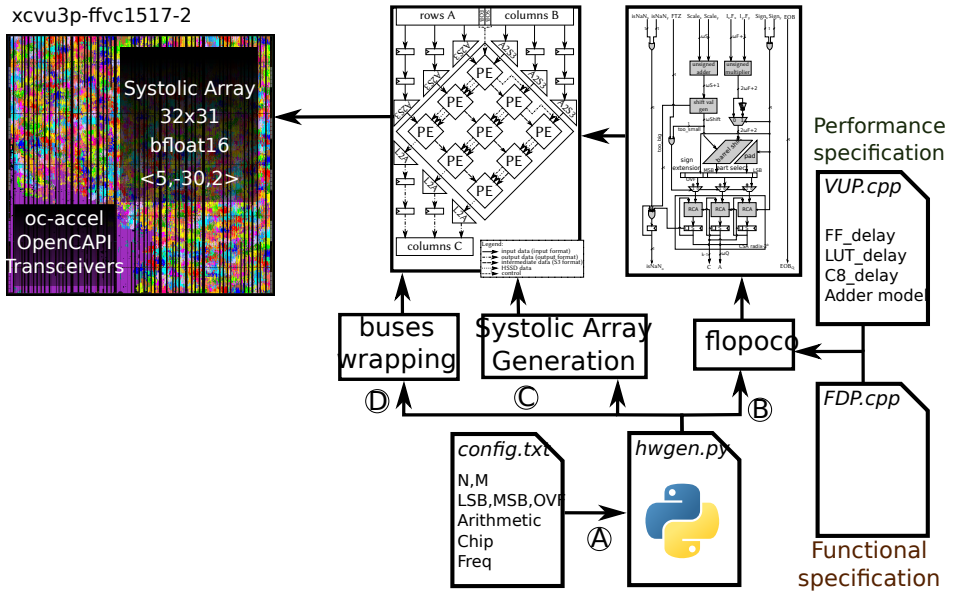
Open Source SW/HW Co-designed Framework



The 2 phases of the framework: right, the a priori Hardware generation flow, and left, the runtime execution flow.

Framework: Hardware Side

- Configuration files to bitstream
- Modified FloPoCo [dDi19]
- OpenCAPI offers ≈23GBp/s with 1024-bit bus @200MHz



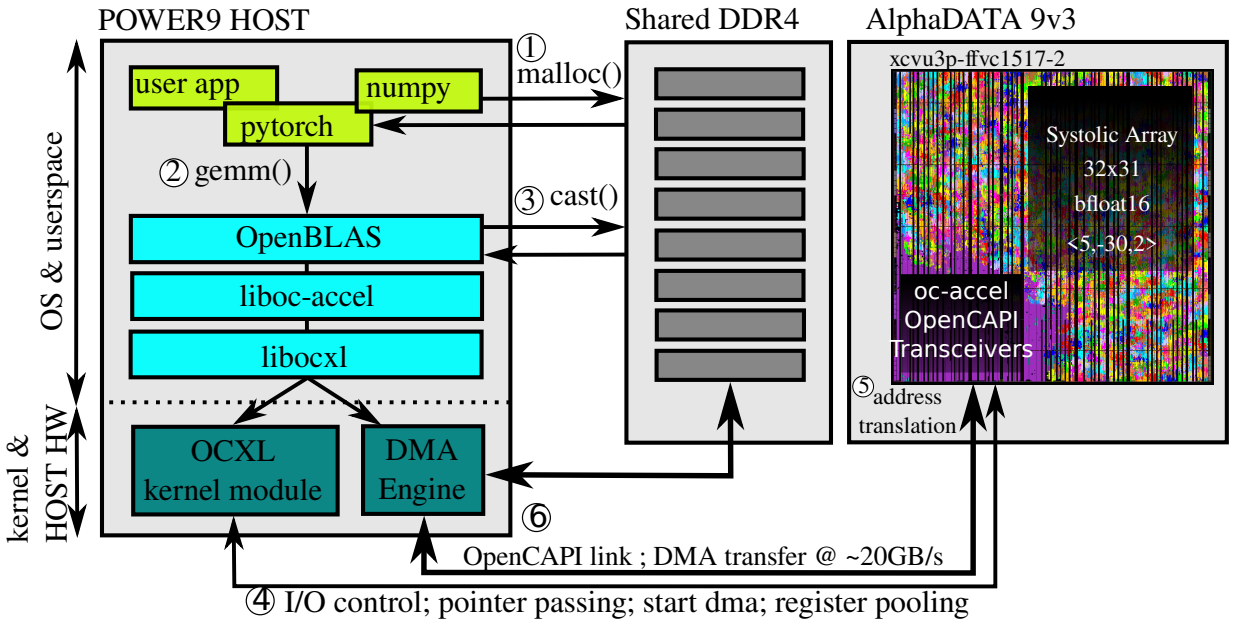
Automated HW generation

[dDi19] de Dinechin, "Reflections on 10 years of FloPoCo" (2019).

Framework: Software Side

Strategy:

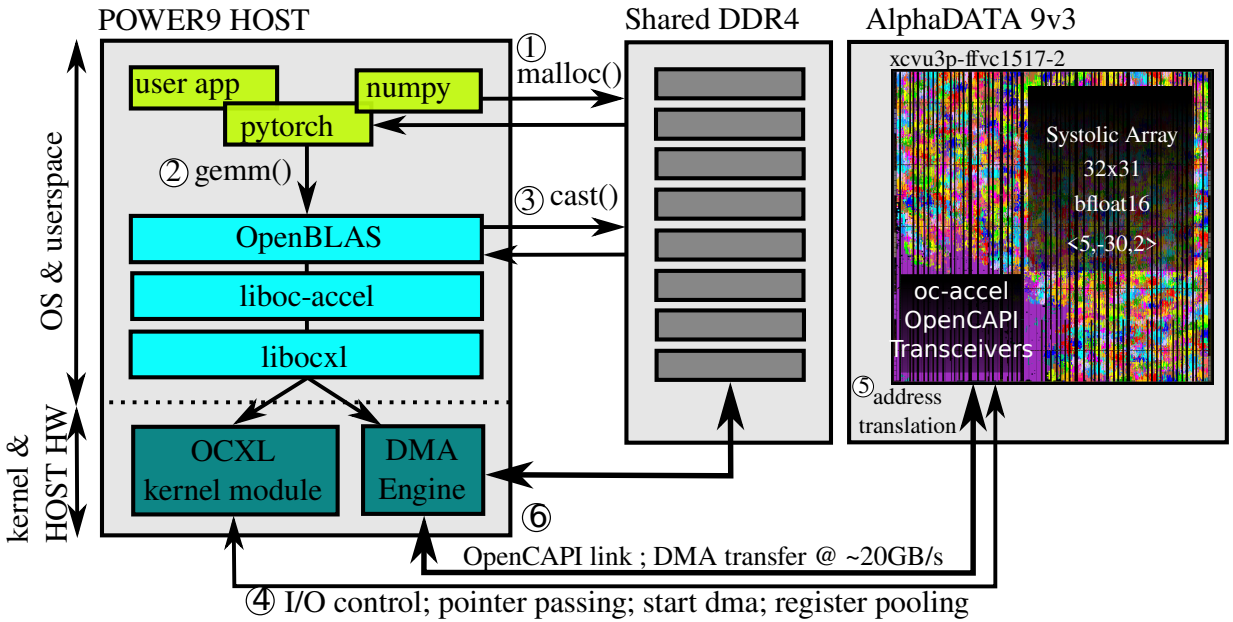
- HPC codes rely on **BLAS calls**
- **Numpy, Pytorch**
- Intercept, cast, dispatch



Software flow

Framework: Software Side

- **User Process:**
  1. Matrix Allocation
  2. Call `gemm()`
- **Our lib:**
  3. Arithmetic cast
  4. Signal DMA properties
  5. HW address translation
  6. DMA starts



Software flow

SW Code Example

```
LD_LIBRARY_PATH=/opt/lib/our_openblas.lib  ./gemm.py
LD_LIBRARY_PATH=/opt/lib/OpenBLAS.lib  ./gemm.py
```

Calling twice the same user application (python) with our BLAS

```
import numpy as np

m = 604
n = 303
k = 100000000

# calls dgemm
A = np.random.random((m,k))
B = np.random.random((k,n))
C = np.matmul(A, B)

# calls sgemm
A = np.random.random((m,k)).astype(np.float32)
B = np.random.random((k,n)).astype(np.float32)
C = np.matmul(A, B)
```

gemm.py: a python program calling GEMMs. No changes in the code.

Current Progress

Numerically-tailored BLAS

4.1 Executive Summary

4.2 Open Source SW/HW Co-designed Framework

4.3 Workload Evaluation

4.4 Conclusions

Evaluation of HPC Codes: Sea Surface Height (SSH)

- Ocean circulation models
- Monitors climate changes [Bar72]
- **Precision-hungry** workload
- Alternating signs and spans over **15 orders of magnitude**
- Can be solved without hardware
- Kahan SCS, DCS [Kah65]
- Extended Software Precision [BB09])

```
for j=128 to 1: # longitude
    for i=1 to 64:  # latitude
        sum = sum + ssh(i,j)
    end
end
print(sum)  # 32.302734375
```

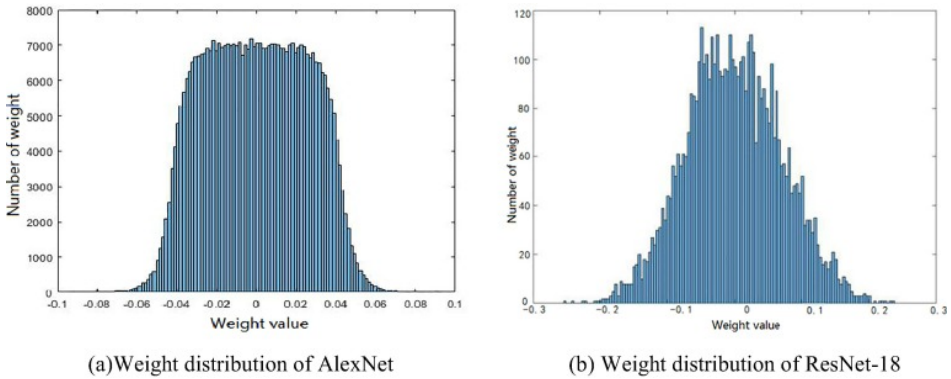
```
for i=64 to 1: # latitude
    for j=1 to 128: # longitude
        sum = sum + ssh(i,j)
    end
end
print(sum)  # 0.734375
```

Double-precision errors.

[Bar72] Barrick, "Remote sensing of sea state by radar" (1972).
[Kah65] Kahan, "Pracniques" (1965).
[BB09] Bailey and Borwein, "High-Precision Computation and Mathematical Physics" (2009).

Evaluation of HPC Codes: Artificial Intelligence

- GEMMs (im2col, FC, Conv)
- **Precision-resilient** workloads [Joh18]
- Accumulations of closely related values [LJZ+21]



(a)Weight distribution of AlexNet (b) Weight distribution of ResNet-18

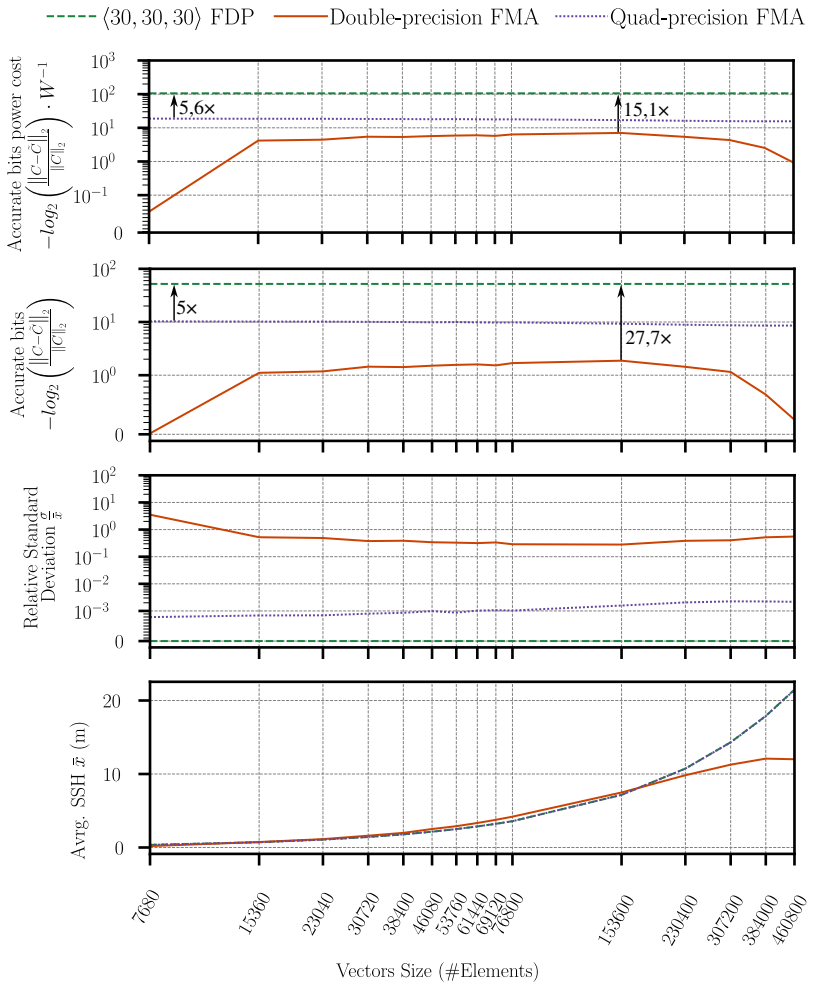
Weights histograms Alexnet/Resnet18

[LJZ+21] Li et al., "Robustness-aware 2-bit quantization with real-time performance for neural network" (2021).

[Joh18] Johnson, "Rethinking floating point for deep learning" (2018).

Evaluation: SSH results

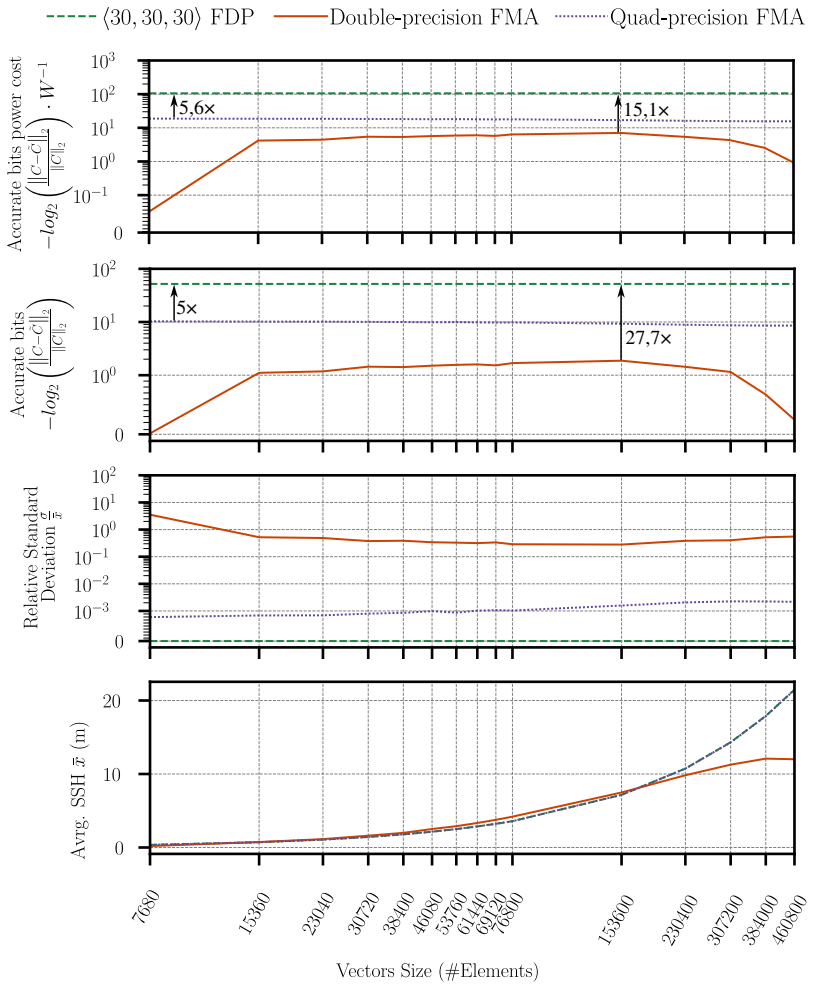
- **IEEE754-64-FDP** vs. **IEEE754-64/128 FMAs**
  - 64-bit format + 91-bit FDP: <ovf:30,msb:30,lsb:-30>
  - 128-bit quad-precision FMA
  - 64-bit double-precision FMA
- **10 shuffles** per chunk size
- **4 metrics**: mean, RSD, Accuracy, Accuracy cost
- The FDP maintains **reproducibility** unlike the FMA
- The FDP always exhibits **52** correct bits
  - **5x** better than quad-precision
  - **27.7x** better than double-precision
- At 200MHz, power consumptions are:
  - 0.266W for IEEE754-64
  - 0.549W for IEEE754-128
  - 0.491W for 91-bit FDP
- For the same Wattage, our FDP generates:
  - **5.6x** more correct bits than IEEE754-128
  - **15.1x** more correct bits than IEEE754-64



IEEE754-64/128 FMAs vs. [64+91]-FDP

Evaluation: SSH results

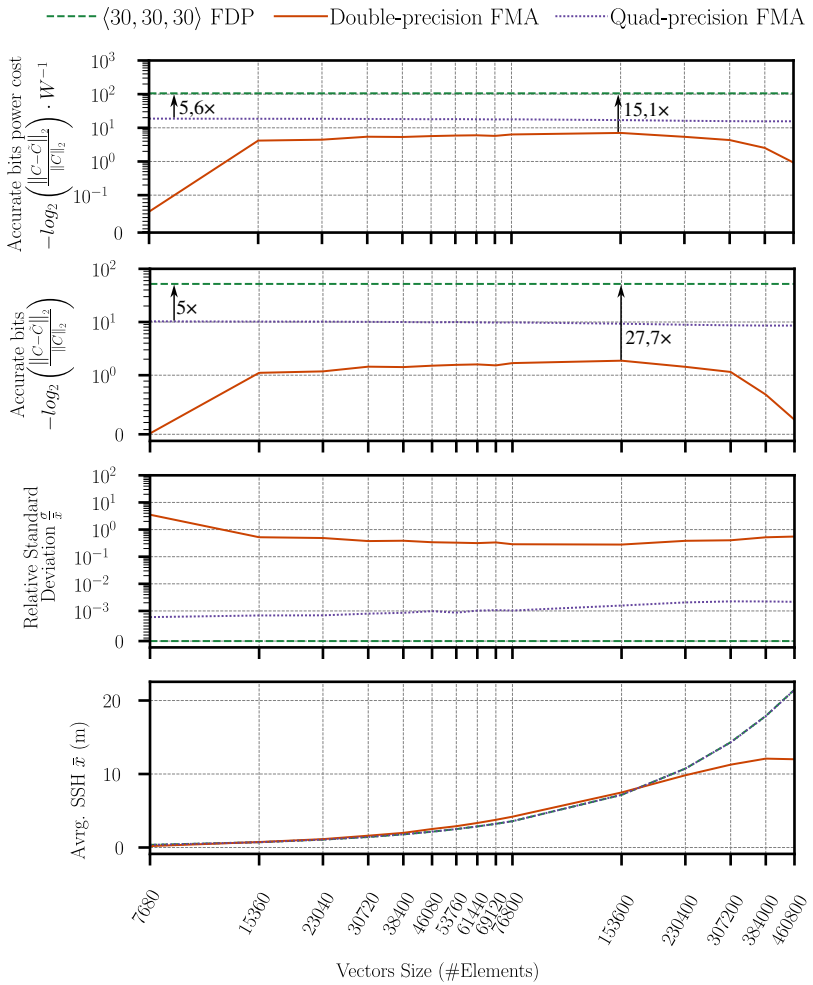
- **IEEE754-64-FDP** vs. **IEEE754-64/128 FMAs**
  - 64-bit format + 91-bit FDP: <ovf:30,msb:30,lsb:-30>
  - 128-bit quad-precision FMA
  - 64-bit double-precision FMA
- **10 shuffles** per chunk size
- **4 metrics**: mean, RSD, Accuracy, Accuracy cost
- The FDP maintains **reproducibility** unlike the FMA
- The FDP always exhibits **52** correct bits
  - **5x** better than quad-precision
  - **27.7x** better than double-precision
- At 200MHz, power consumptions are:
  - 0.266W for IEEE754-64
  - 0.549W for IEEE754-128
  - 0.491W for 91-bit FDP
- For the same Wattage, our FDP generates:
  - **5.6x** more correct bits than IEEE754-128
  - **15.1x** more correct bits than IEEE754-64



IEEE754-64/128 FMAs vs. [64+91]-FDP

Evaluation: SSH results

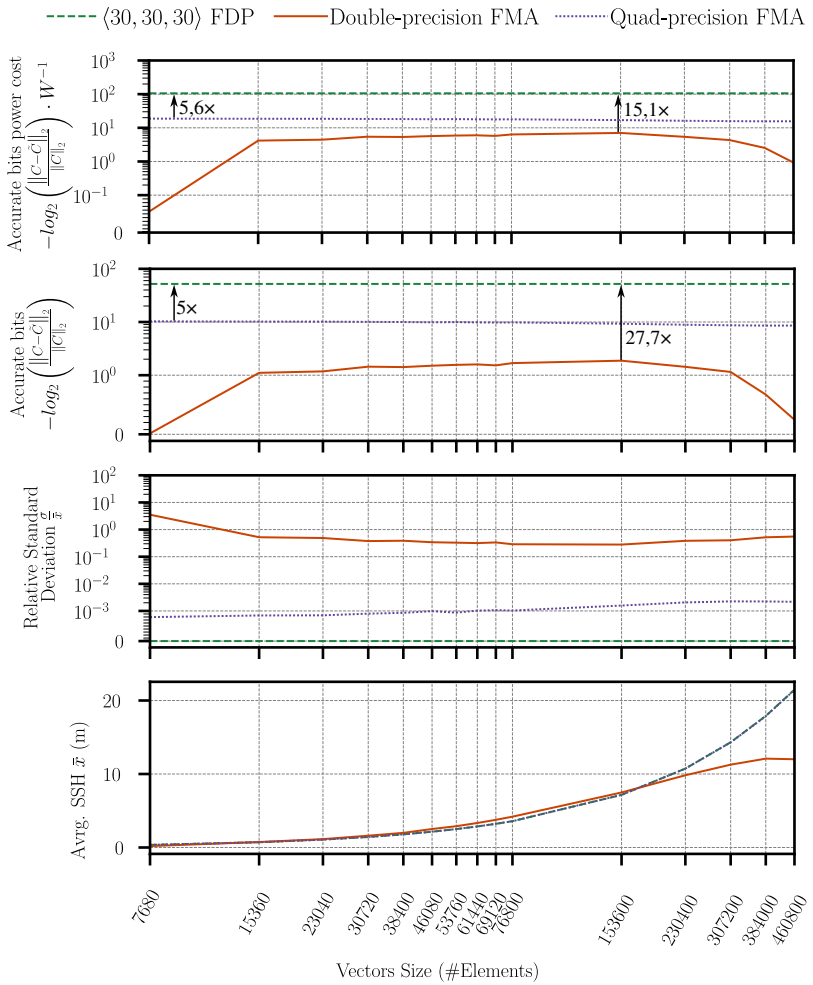
- **IEEE754-64-FDP** vs. **IEEE754-64/128 FMAs**
  - 64-bit format + 91-bit FDP: <ovf:30,msb:30,lsb:-30>
  - 128-bit quad-precision FMA
  - 64-bit double-precision FMA
- **10 shuffles** per chunk size
- **4 metrics**: mean, RSD, Accuracy, Accuracy cost
- The FDP maintains **reproducibility** unlike the FMA
- The FDP always exhibits **52** correct bits
  - **5x** better than quad-precision
  - **27.7x** better than double-precision
- At 200MHz, power consumptions are:
  - 0.266W for IEEE754-64
  - 0.549W for IEEE754-128
  - 0.491W for 91-bit FDP
- For the same Wattage, our FDP generates:
  - **5.6x** more correct bits than IEEE754-128
  - **15.1x** more correct bits than IEEE754-64



IEEE754-64/128 FMAs vs. [64+91]-FDP

Evaluation: SSH results

- **IEEE754-64-FDP** vs. **IEEE754-64/128 FMAs**
  - 64-bit format + 91-bit FDP: <ovf:30,msb:30,lsb:-30>
  - 128-bit quad-precision FMA
  - 64-bit double-precision FMA
- **10 shuffles** per chunk size
- **4 metrics**: mean, RSD, Accuracy, Accuracy cost
- The FDP maintains **reproducibility** unlike the FMA
- The FDP always exhibits **52** correct bits
  - **5x** better than quad-precision
  - **27.7x** better than double-precision
- At 200MHz, power consumptions are:
  - 0.266W for IEEE754-64
  - 0.549W for IEEE754-128
  - 0.491W for 91-bit FDP
- For the same Wattage, our FDP generates:
  - **5.6x** more correct bits than IEEE754-128
  - **15.1x** more correct bits than IEEE754-64



IEEE754-64/128 FMAs vs. [64+91]-FDP

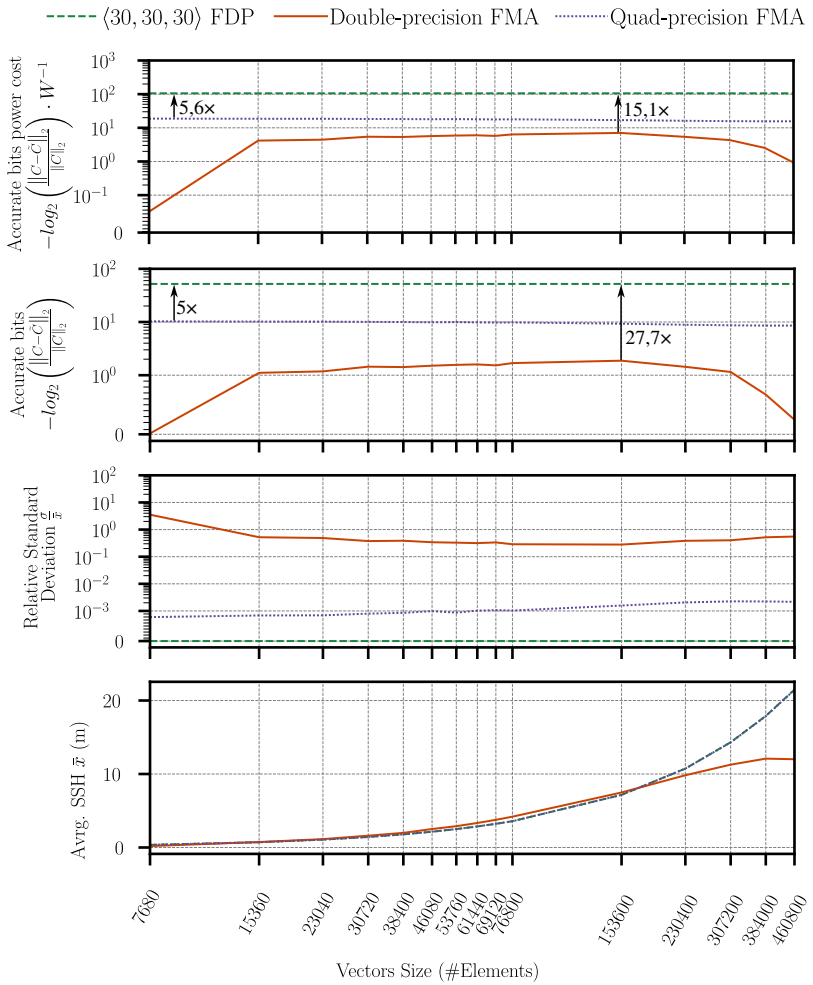
Evaluation: SSH results

- **IEEE754-64-FDP** vs. **IEEE754-64/128 FMAs**
  - 64-bit format + 91-bit FDP: <ovf:30,msb:30,lsb:-30>
  - 128-bit quad-precision FMA
  - 64-bit double-precision FMA
- **10 shuffles** per chunk size
- **4 metrics**: mean, RSD, Accuracy, Accuracy cost
- The FDP maintains **reproducibility** unlike the FMA

- The FDP always exhibits **52** correct bits
  - **5x** better than quad-precision
  - **27.7x** better than double-precision

- At 200MHz, power consumptions are:
  - 0.266W for IEEE754-64
  - 0.549W for IEEE754-128
  - 0.491W for 91-bit FDP

- For the same Wattage, our FDP generates:
  - **5.6x** more correct bits than IEEE754-128
  - **15.1x** more correct bits than IEEE754-64

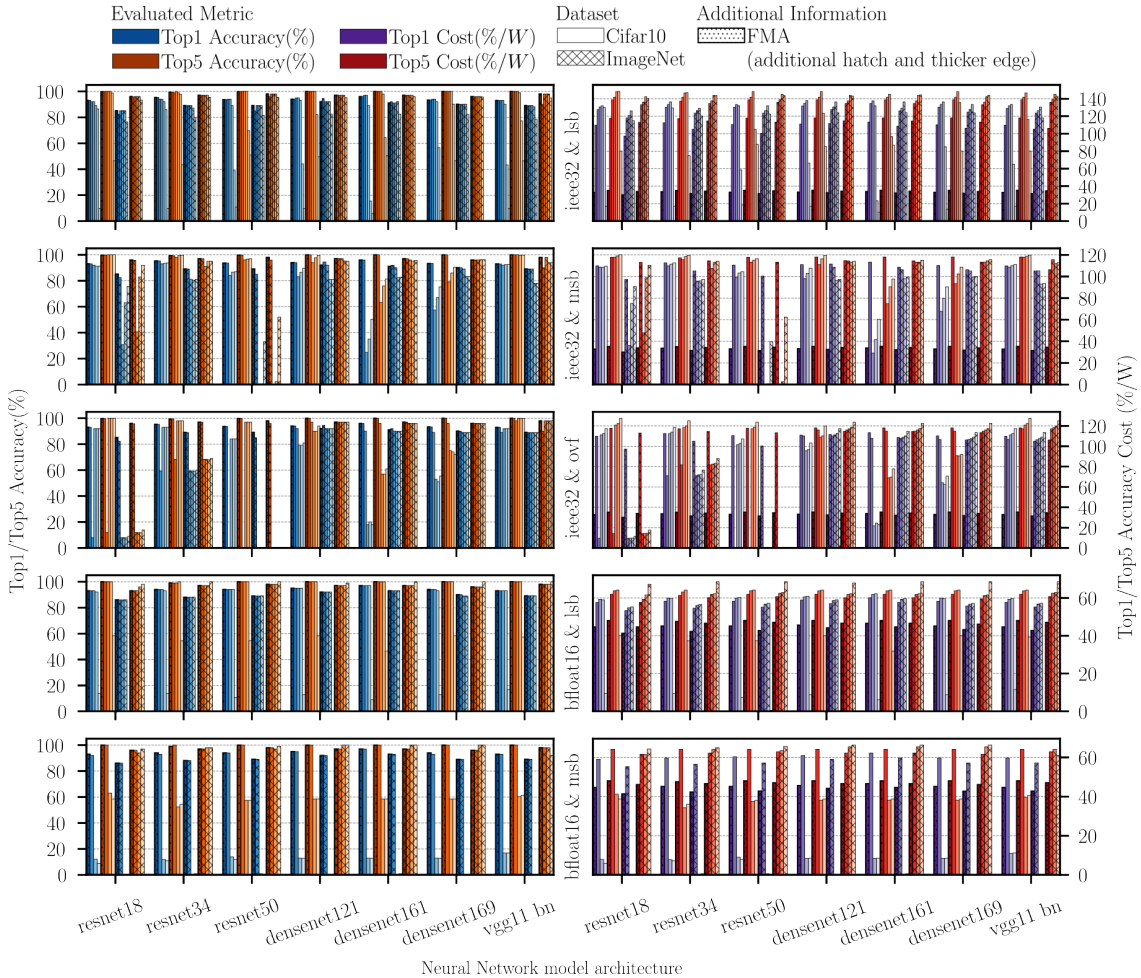


IEEE754-64/128 FMAs vs. [64+91]-FDP

Evaluation: AI results

- **Pytorch code without any changes**
- Top1/5 scores and **"score costs"**

- Evaluated models:
  - ResNet18
  - ResNet34
  - ResNet50
  - DenseNet121
  - DenseNet169
  - VGG11_BN
- Evaluated datasets:
  - CIFAR10
  - ImageNet
- Evaluated computer formats:
  - IEEE754-32
  - BrainFloat16
- 27 accumulators and conventional FMA
- A total of 384 combinations

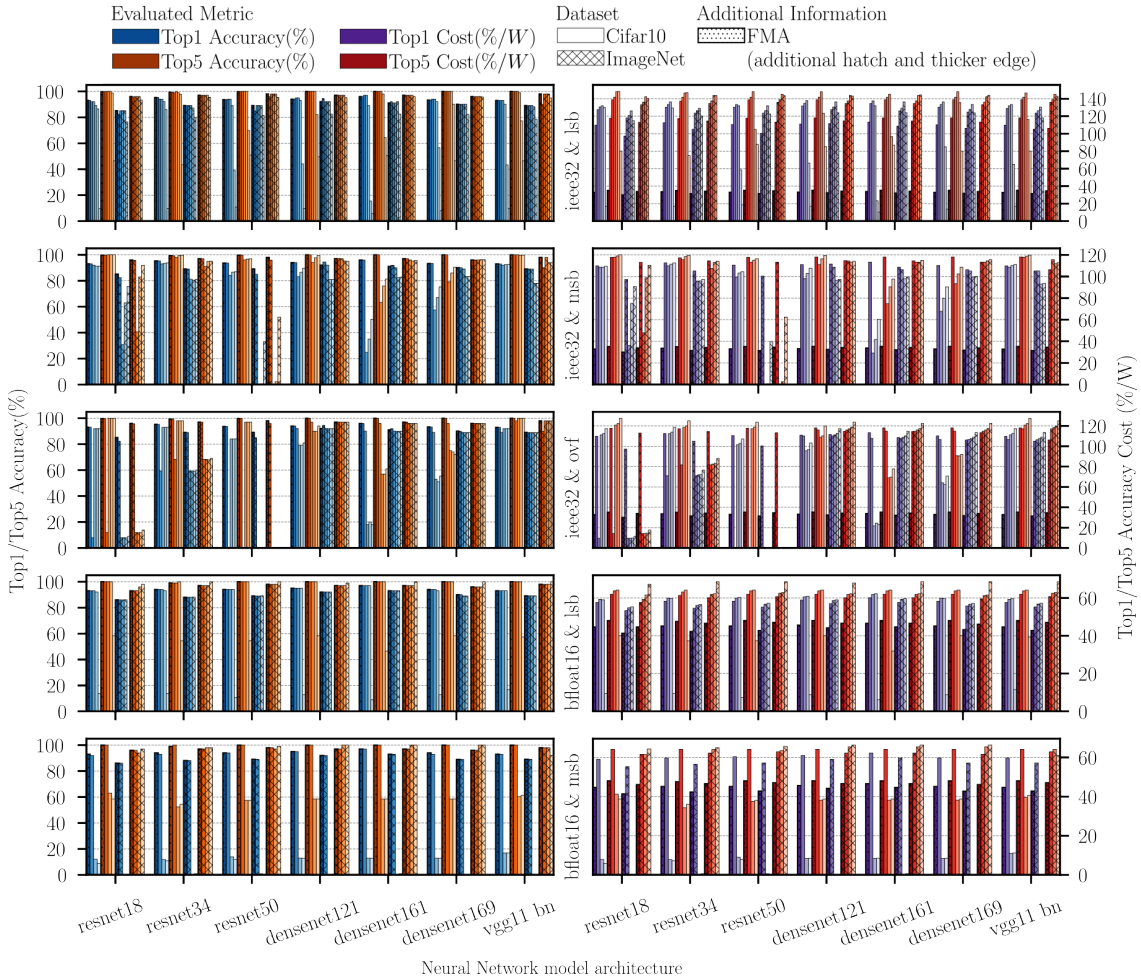


Top1/5 scores/costs across different models/formats/datasets/accumulators

Evaluation: AI results

- **Pytorch code without any changes**
- Top1/5 scores and **"score costs"**

- Evaluated models:
  - ResNet18
  - ResNet34
  - ResNet50
  - DenseNet121
  - DenseNet169
  - VGG11_BN
- Evaluated datasets:
  - CIFAR10
  - ImageNet
- Evaluated computer formats:
  - IEEE754-32
  - BrainFloat16
- 27 accumulators and conventional FMA
- A total of 384 combinations

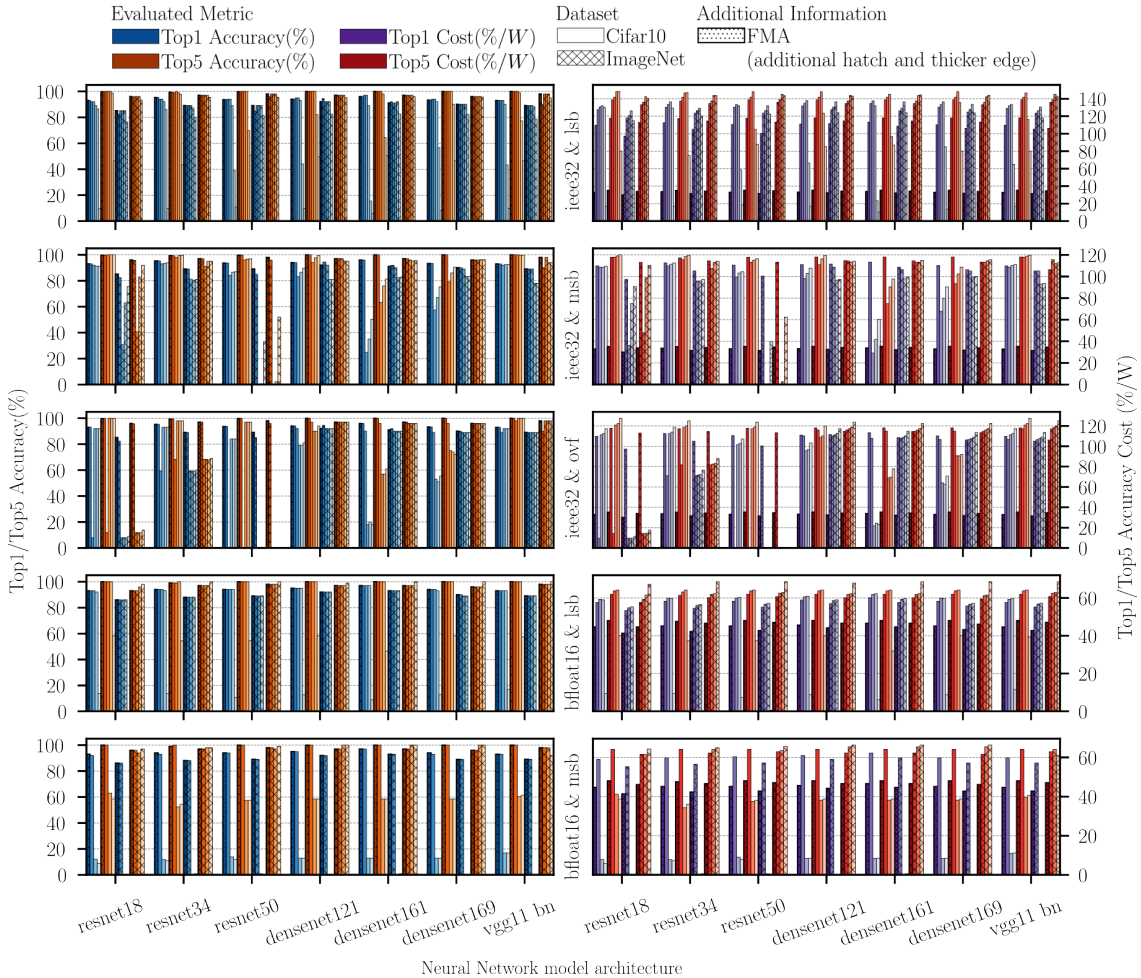


Top1/5 scores/costs across different models/formats/datasets/accumulators

Evaluation: AI results

- **Pytorch code without any changes**
- Top1/5 scores and **"score costs"**

- Evaluated models:
  - ResNet18
  - ResNet34
  - ResNet50
  - DenseNet121
  - DenseNet169
  - VGG11_BN
- Evaluated datasets:
  - CIFAR10
  - ImageNet
- Evaluated computer formats:
  - IEEE754-32
  - BrainFloat16
- 27 accumulators and conventional FMA
- A total of **384** combinations

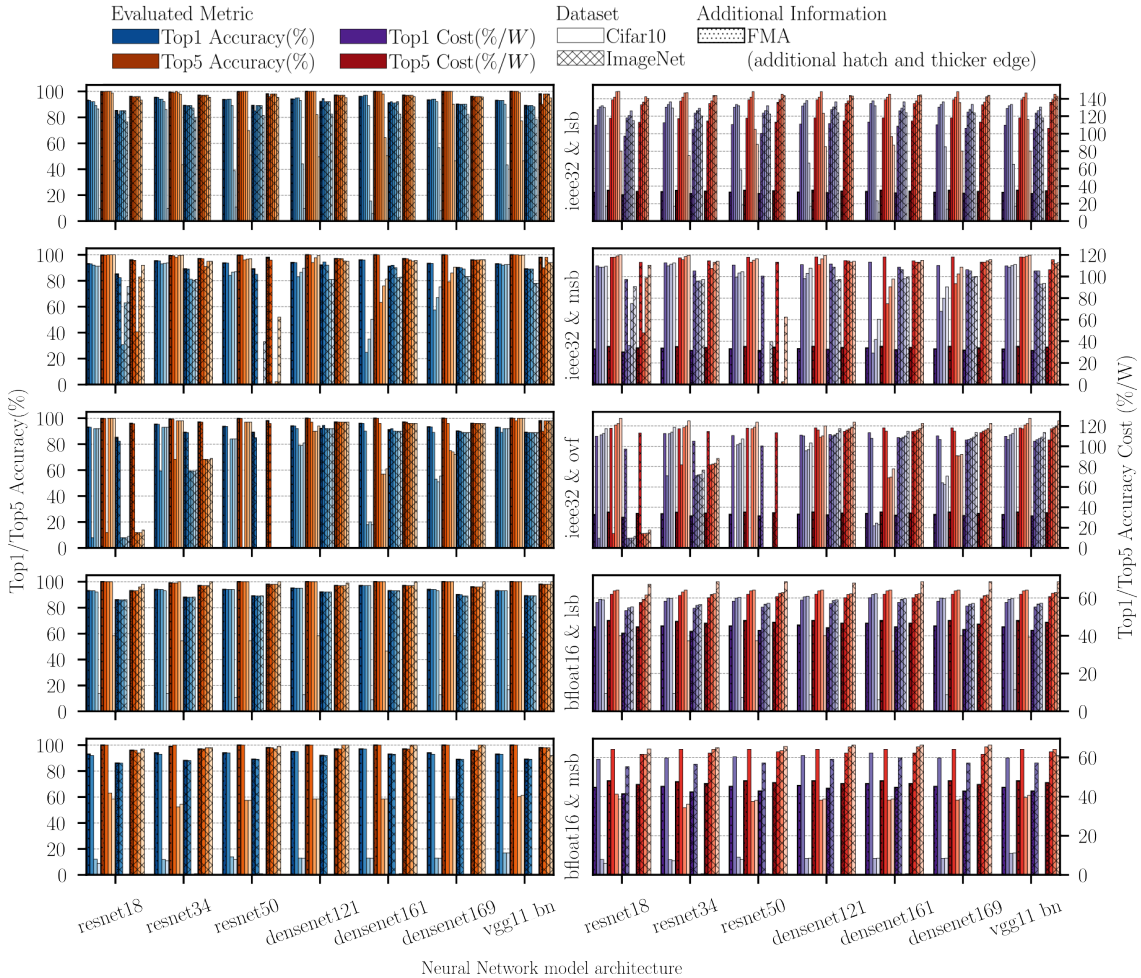


Top1/5 scores/costs across different models/formats/datasets/accumulators

Evaluation: AI results

- **Pytorch code without any changes**
- Top1/5 scores and **"score costs"**

- Evaluated models:
  - ResNet18
  - ResNet34
  - ResNet50
  - DenseNet121
  - DenseNet169
  - VGG11_BN
- Evaluated datasets:
  - CIFAR10
  - ImageNet
- Evaluated computer formats:
  - IEEE754-32
  - BrainFloat16
- 27 accumulators and conventional FMA
- A total of **384** combinations

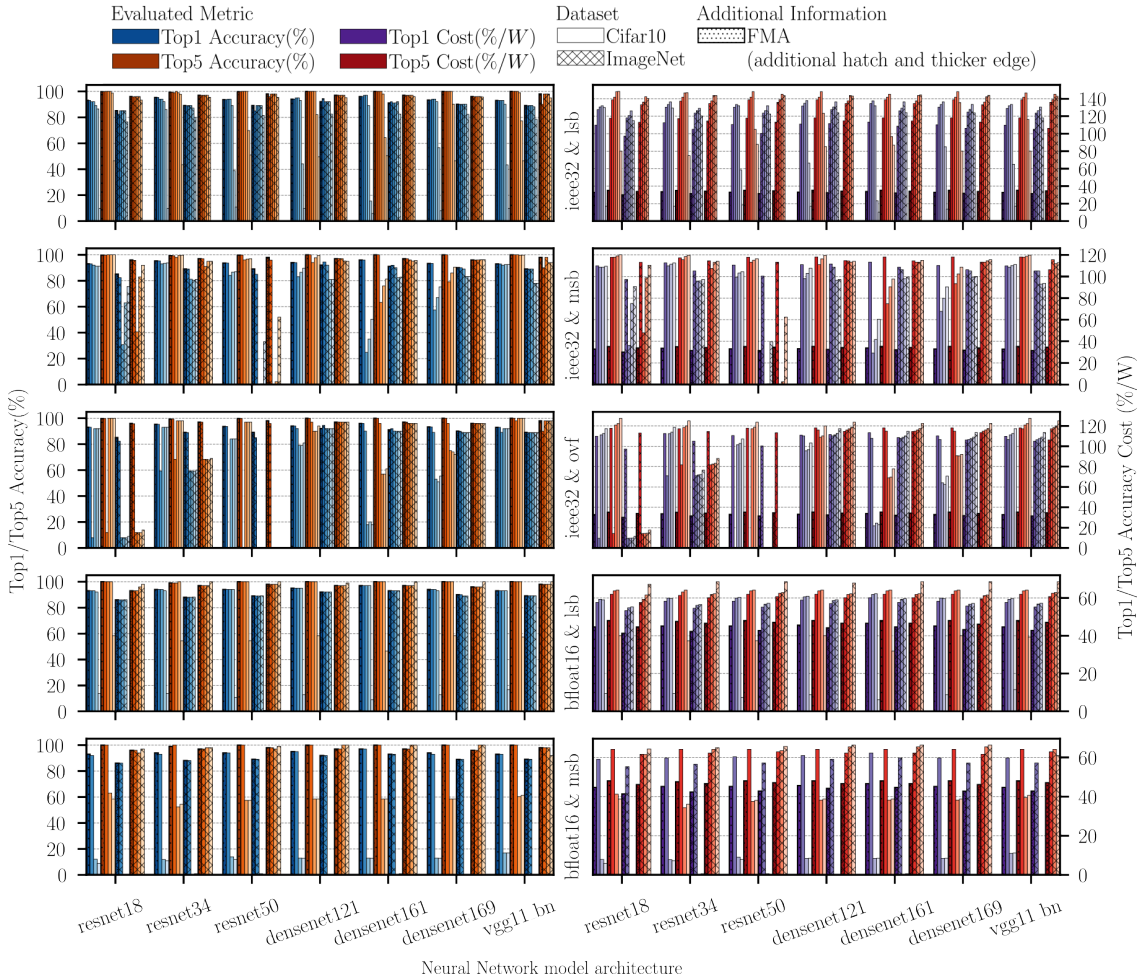


Top1/5 scores/costs across different models/formats/datasets/accumulators

Evaluation: AI results

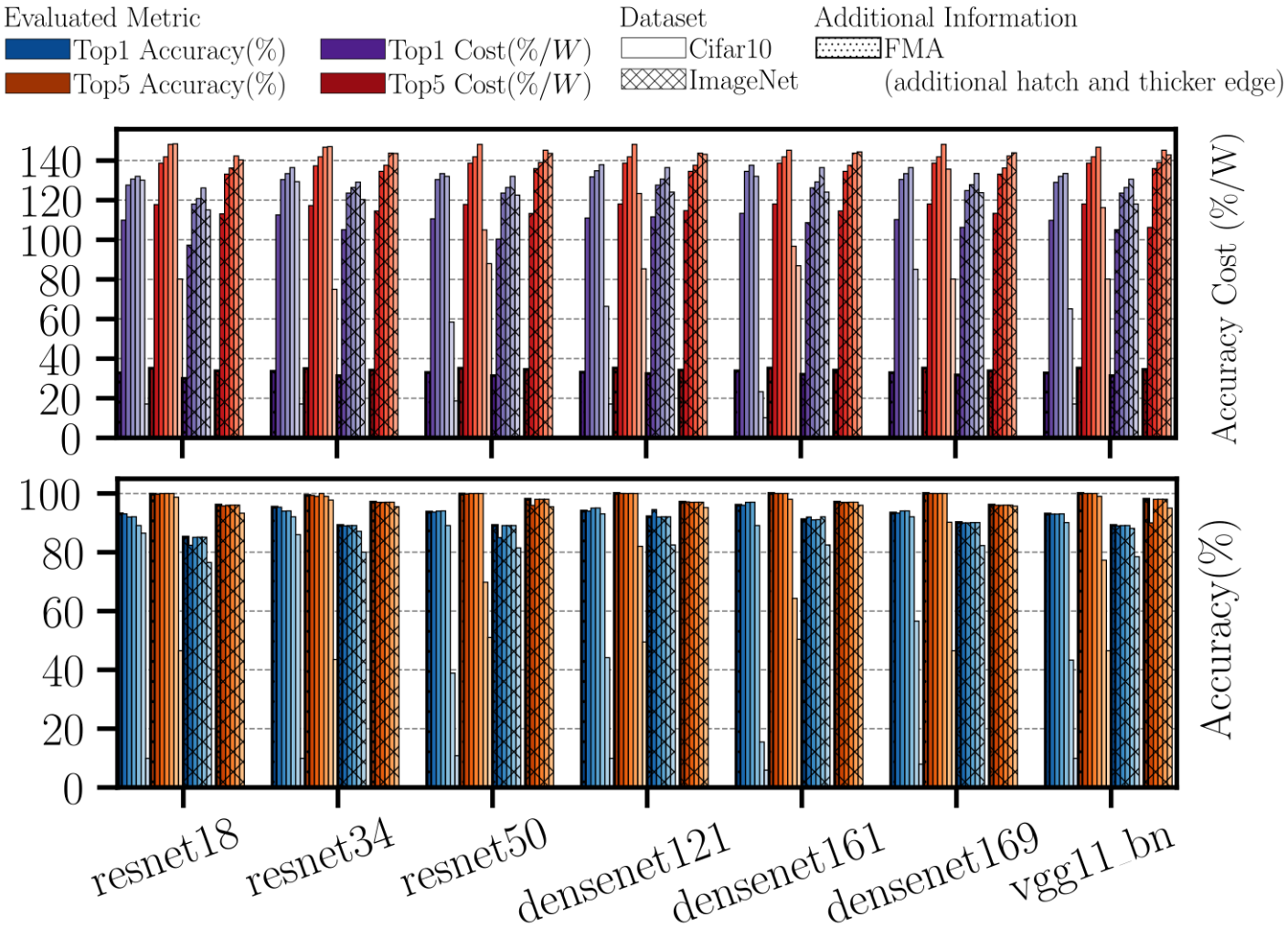
- **Pytorch code without any changes**
- Top1/5 scores and **"score costs"**

- Evaluated models:
  - ResNet18
  - ResNet34
  - ResNet50
  - DenseNet121
  - DenseNet169
  - VGG11_BN
- Evaluated datasets:
  - CIFAR10
  - ImageNet
- Evaluated computer formats:
  - IEEE754-32
  - BrainFloat16
- 27 accumulators and conventional FMA
- A total of **384** combinations



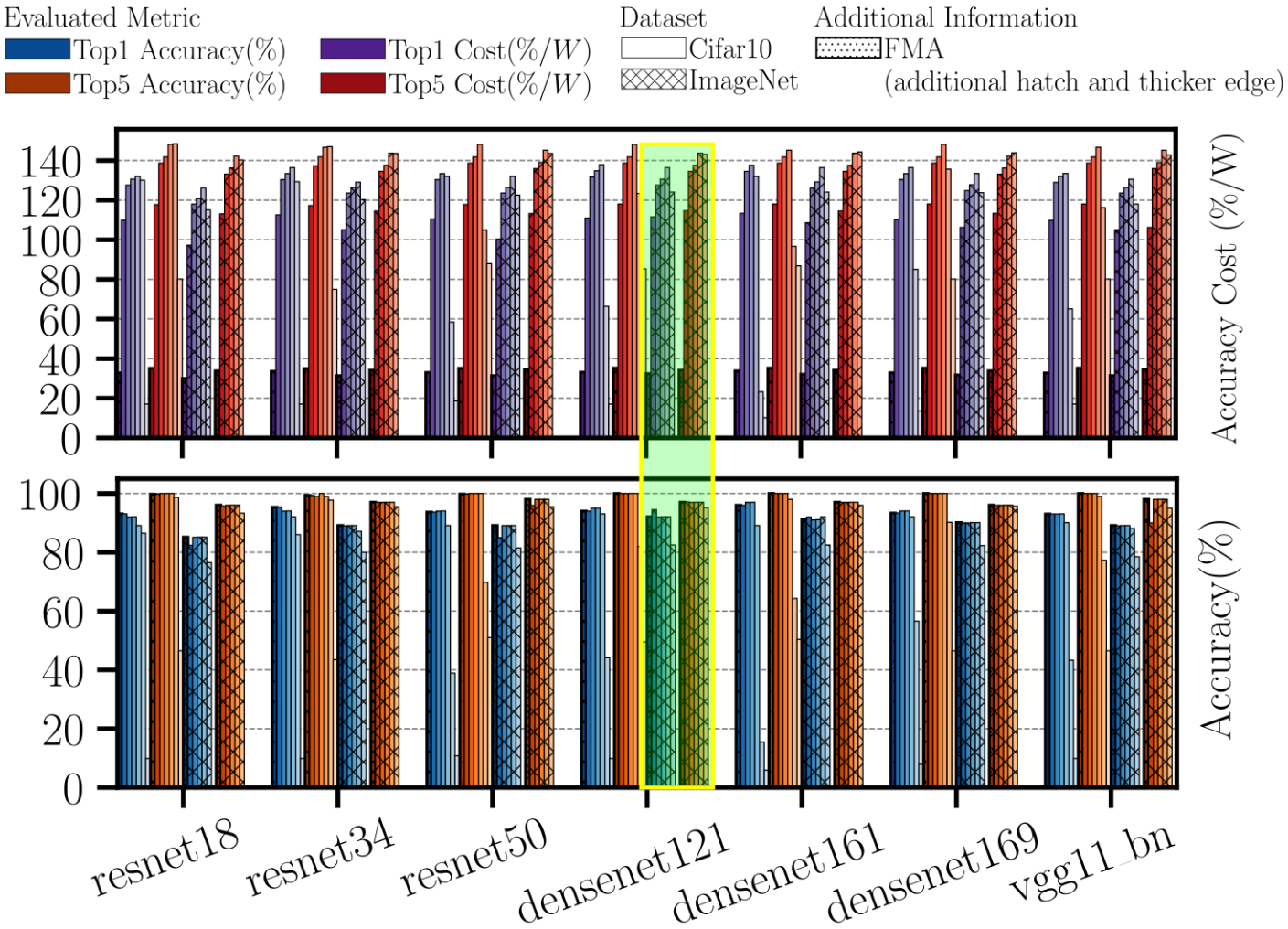
Top1/5 scores/costs across different models/formats/datasets/accumulators

A zoom on the LSB case



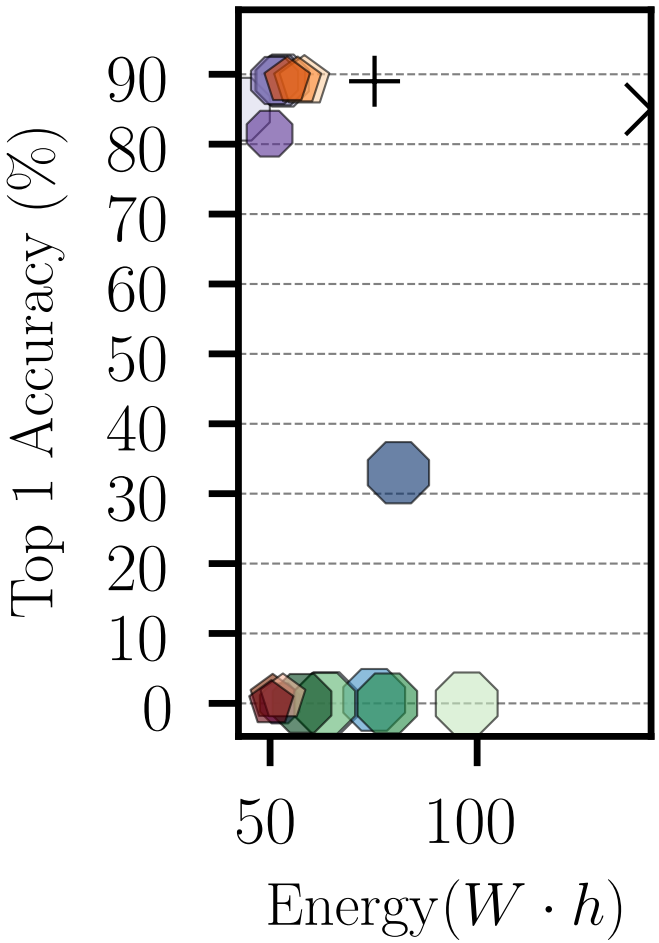
The impact of lowering number of LSBs on Top1/5 scores and score costs.

A zoom on the LSB + densenet121 + ImageNet case



The impact of lowering number of LSBs on Top1/5 scores and score costs.

Evaluation: AI results



(Comp. format, accum. param.)

- (IEEE32,LSB)
- (IEEE32,MSB)
- (IEEE32,OVF)
- (Bfloat16,LSB)
- (Bfloat16,MSB)

Accumulator Size

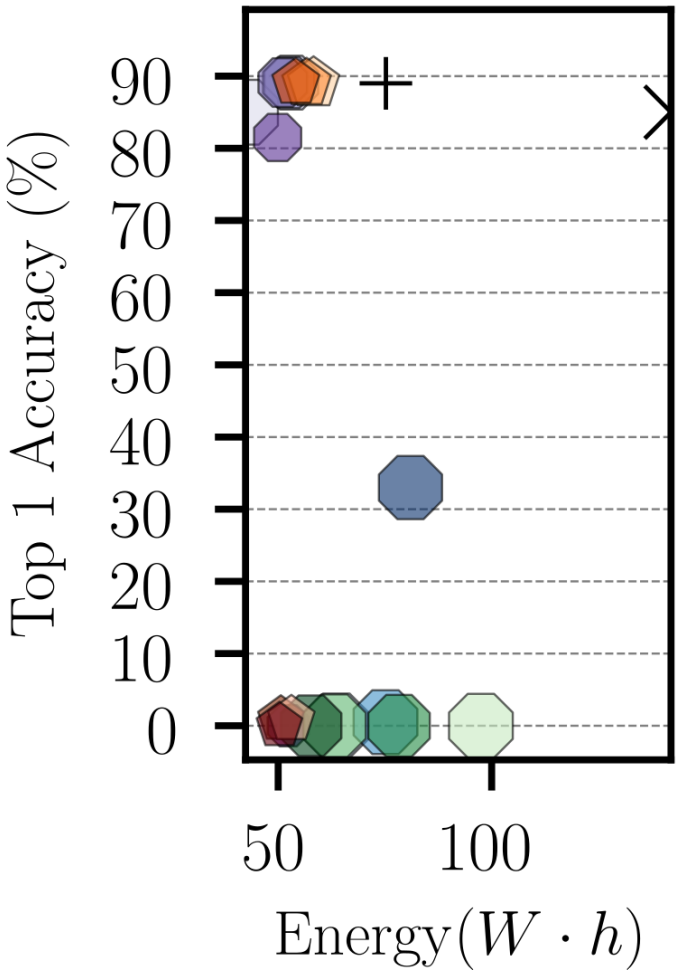
- 10 bits
- 20 bits
- 30 bits
- 40 bits
- 50 bits

Conventional HW comparison

- × IEEE754-32 FMA
- \+ Bfloat16 FMA

Top1 scores VS. Energy cost of inferencing validation set

Zoom on ImageNet + Resnet50



(Comp. format, accum. param.)

- (IEEE32,LSB)
- (IEEE32,MSB)
- (IEEE32,OVF)
- (Bfloat16,LSB)
- (Bfloat16,MSB)

Accumulator Size

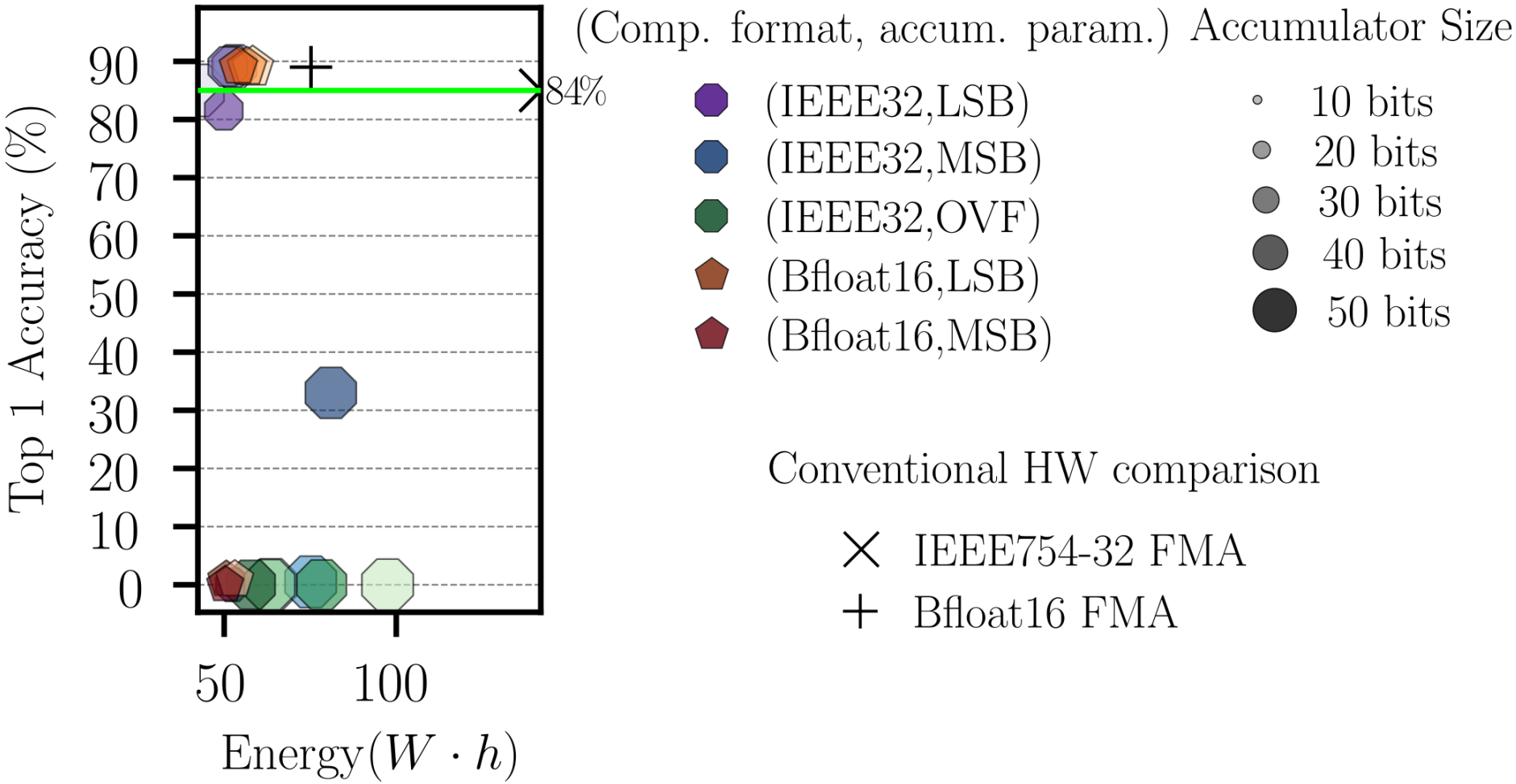
- 10 bits
- 20 bits
- 30 bits
- 40 bits
- 50 bits

Conventional HW comparison

- × IEEE754-32 FMA
- \+ Bfloat16 FMA

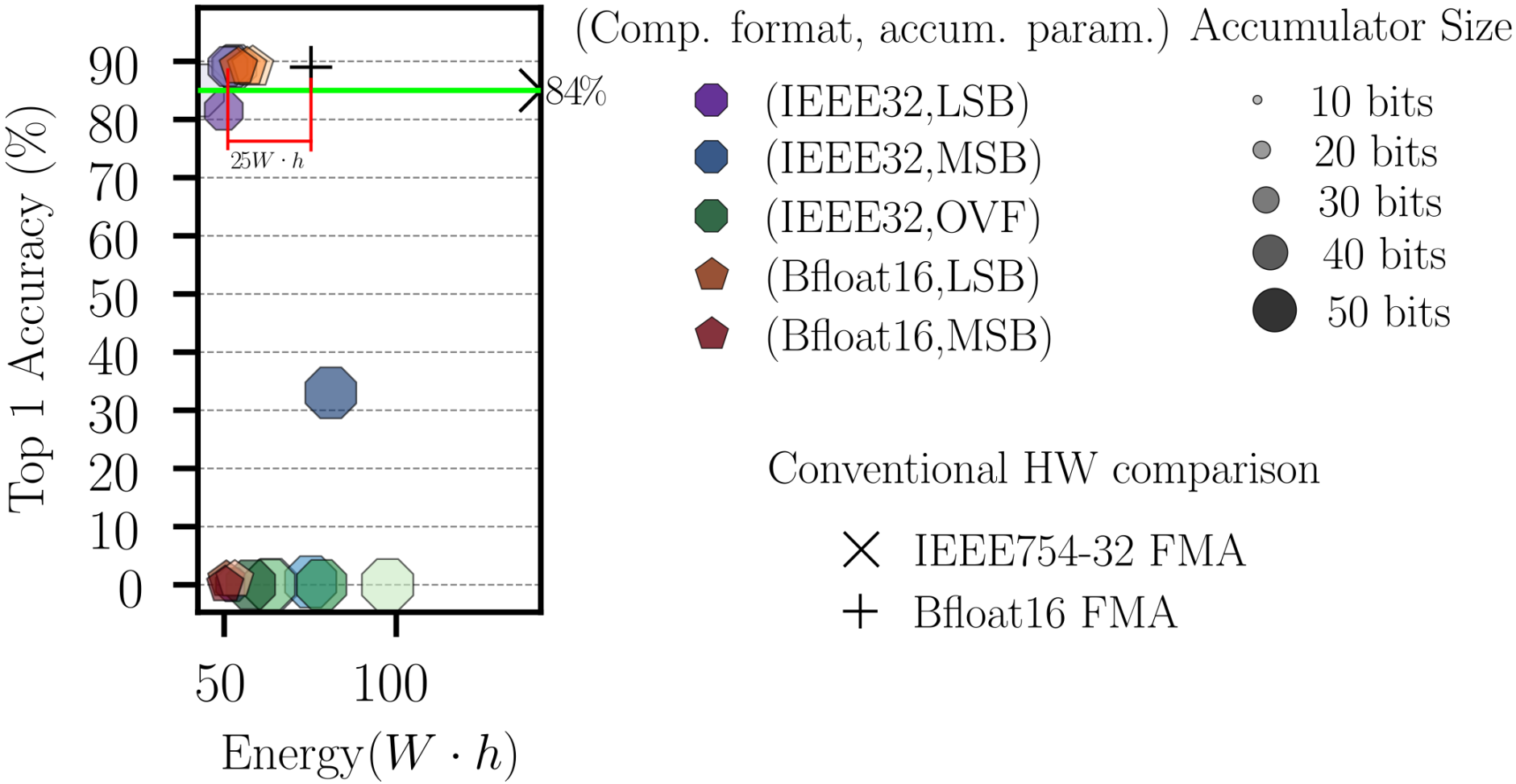
Test Set Inference Cost VS. Top1 Score.

Zoom on ImageNet + Resnet50



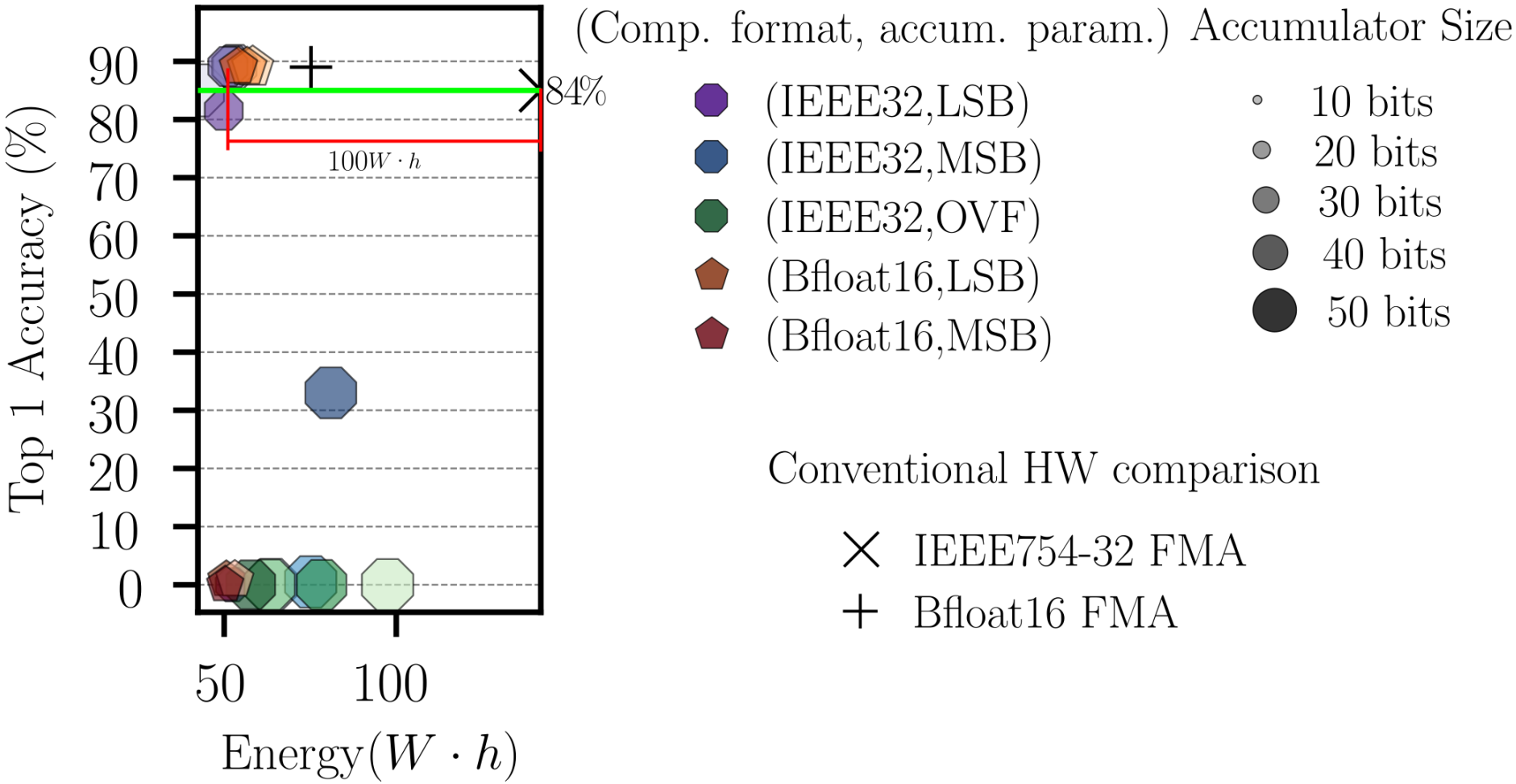
Test Set Inference Cost VS. Top1 Score.

Zoom on ImageNet + Resnet50



Test Set Inference Cost VS. Top1 Score.

Zoom on ImageNet + Resnet50



Test Set Inference Cost VS. Top1 Score.

Zoom on ImageNet + Resnet50

Graphical Superiority

Graphically, it always exists one **FDP** to the left (indicating lower energy) and are never below (ensuring equal or greater accuracy) conventional FMAs, **highlighting their superior performance**

Current Progress

Numerically-tailored BLAS

4.1 Executive Summary

4.2 Open Source SW/HW Co-designed Framework

4.3 Workload Evaluation

4.4 Conclusions

Summary of Insights and Key Contributions

Open-Source Framework

A **bridge** from user applications to silicon, focusing on **tailored arithmetic** solutions.

Transparent Integration

Seamlessly integrates **without code modification**, agnostic of underlying hardware.

Enhancing State-of-the-Art

Improves **energy efficiency and accuracy** across **drastically diverse HPC applications**.

Findings

System-wise Arithmetic and numerically-tailored solutions alleviate the power wall (A), and the concern of workload sparsity (B).

Summary of Insights and Key Contributions

Open-Source Framework

A **bridge** from user applications to silicon, focusing on **tailored arithmetic** solutions.

Transparent Integration

Seamlessly integrates **without code modification**, agnostic of underlying hardware.

Enhancing State-of-the-Art

Improves **energy efficiency and accuracy** across **drastically diverse HPC applications**.

Findings

System-wise Arithmetic and numerically-tailored solutions alleviate the power wall (A), and the concern of workload sparsity (B).

Summary of Insights and Key Contributions

Open-Source Framework

A **bridge** from user applications to silicon, focusing on **tailored arithmetic** solutions.

Transparent Integration

Seamlessly integrates **without code modification**, agnostic of underlying hardware.

Enhancing State-of-the-Art

Improves **energy efficiency and accuracy** across **drastically diverse HPC applications**.

Findings

System-wise Arithmetic and numerically-tailored solutions alleviate the power wall (A), and the concern of workload sparsity (B).

Summary of Insights and Key Contributions

Open-Source Framework

A **bridge** from user applications to silicon, focusing on **tailored arithmetic** solutions.

Transparent Integration

Seamlessly integrates **without code modification**, agnostic of underlying hardware.

Enhancing State-of-the-Art

Improves **energy efficiency and accuracy** across **drastically diverse HPC applications**.

Findings

System-wise Arithmetic and numerically-tailored solutions alleviate the power wall (A), and the concern of workload sparsity (B).

Current Progress

"Divisions in Vector Processing: Leveraging Opportunities from a Paradigm Shift"

Floating-Point Divisions

5.1 Executive Summary

5.2 Algorithmic Contribution

5.3 Results

5.4 Conclusions

Executive Summary: Vector Processing Revival

Opportunity 1: Vector Processing

Resurgence of **vector processing** exemplified by the **RISC-V "V"** extension. **Enhance Data-level parallelism (DLP)** by trading **space and time** implementations.

Opportunity 2: Open Source Silicon

The momentum in **open-source silicon community** facilitates **tape-outs in academia**.

Algorithmic Contribution: Division Algorithm

Less famous operation but useful: **N-body simulation**; proposal: **muliple slower, area/power-efficient** units.

Open ASIC Flow Contribution

Custom Open ASIC flow for **extensive design space exploration** to characterize the divisions.

Executive Summary: Vector Processing Revival

Opportunity 1: Vector Processing

Resurgence of **vector processing** exemplified by the **RISC-V "V"** extension. **Enhance Data-level parallelism (DLP)** by trading **space and time** implementations.

Opportunity 2: Open Source Silicon

The momentum in **open-source silicon community** facilitates **tape-outs in academia**.

Algorithmic Contribution: Division Algorithm

Less famous operation but useful: **N-body simulation**; proposal: **muliple slower, area/power-efficient** units.

Open ASIC Flow Contribution

Custom Open ASIC flow for **extensive design space exploration** to characterize the divisions.

Executive Summary: Vector Processing Revival

Opportunity 1: Vector Processing

Resurgence of **vector processing** exemplified by the **RISC-V "V"** extension. **Enhance Data-level parallelism (DLP)** by trading **space and time** implementations.

Opportunity 2: Open Source Silicon

The momentum in **open-source silicon community** facilitates **tape-outs in academia**.

Algorithmic Contribution: Division Algorithm

Less famous operation but useful: **N-body simulation**; proposal: **muliple slower, area/power-efficient** units.

Open ASIC Flow Contribution

Custom Open ASIC flow for **extensive design space exploration** to characterize the divisions.

Executive Summary: Vector Processing Revival

Opportunity 1: Vector Processing

Resurgence of **vector processing** exemplified by the **RISC-V "V"** extension. **Enhance Data-level parallelism (DLP)** by trading **space and time** implementations.

Opportunity 2: Open Source Silicon

The momentum in **open-source silicon community** facilitates **tape-outs in academia**.

Algorithmic Contribution: Division Algorithm

Less famous operation but useful: **N-body simulation**; proposal: **muliple slower, area/power-efficient** units.

Open ASIC Flow Contribution

Custom Open ASIC flow for **extensive design space exploration** to characterize the divisions.

Current Progress

Floating-Point Divisions

5.1 Executive Summary

5.2 Algorithmic Contribution

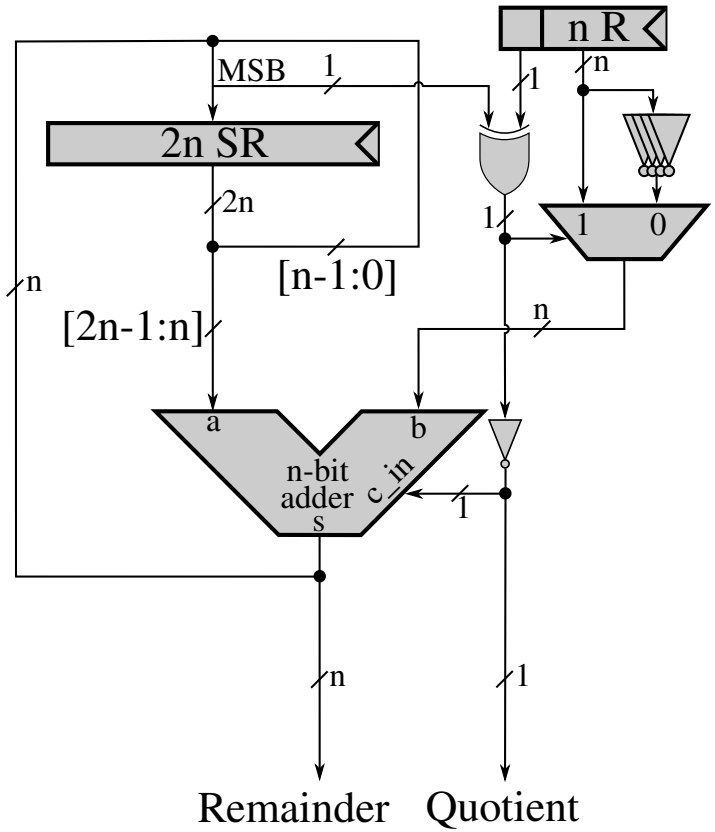
5.3 Results

5.4 Conclusions

Algorithmic Proposal

Baseline, n the mantissa size:

- Non-branching with a single n-bit adder.
- n iteration.



Non-restoring division circuit

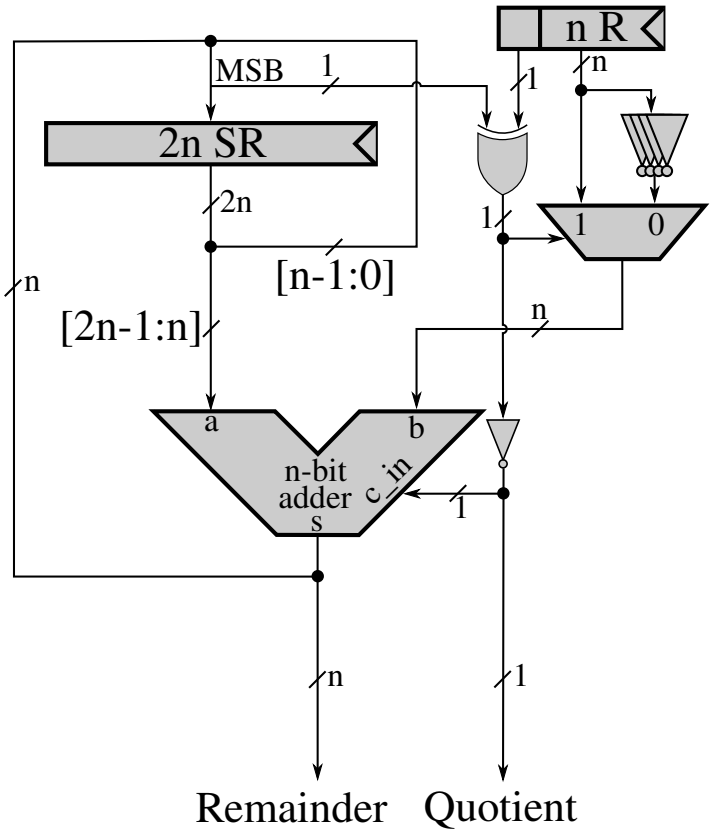
Algorithmic Proposal

Baseline, n the mantissa size:

- Non-branching with a single n-bit adder.
- n iteration.

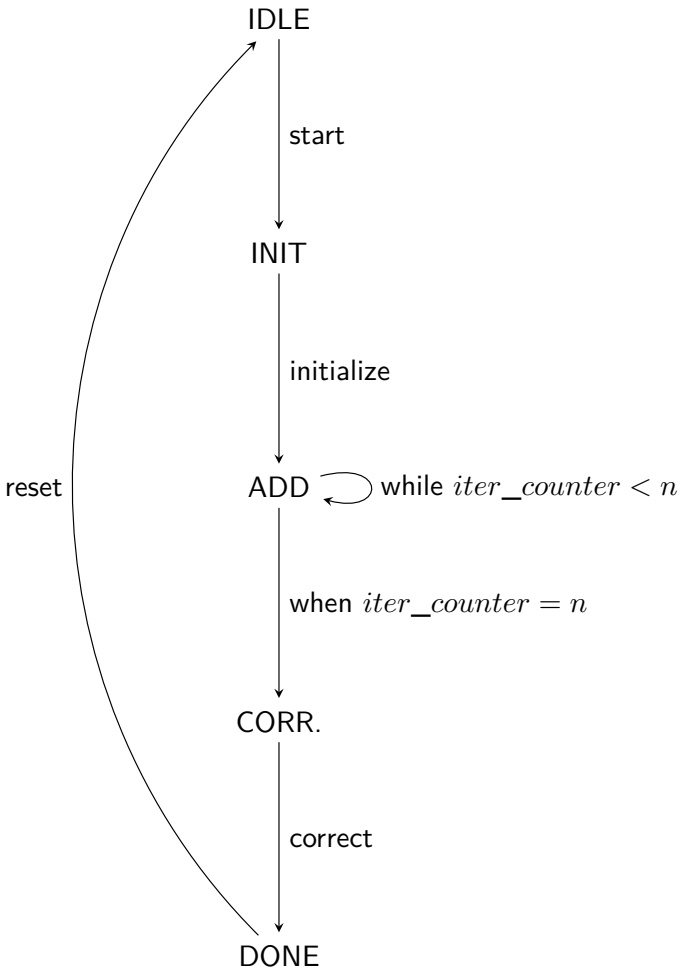
Unrolling Proposal:

- Serial addition of k bits
- $k \in [1; n]$
- Reduces a **power hungry** section
- Reduces the **area**
- Introduces slowdown

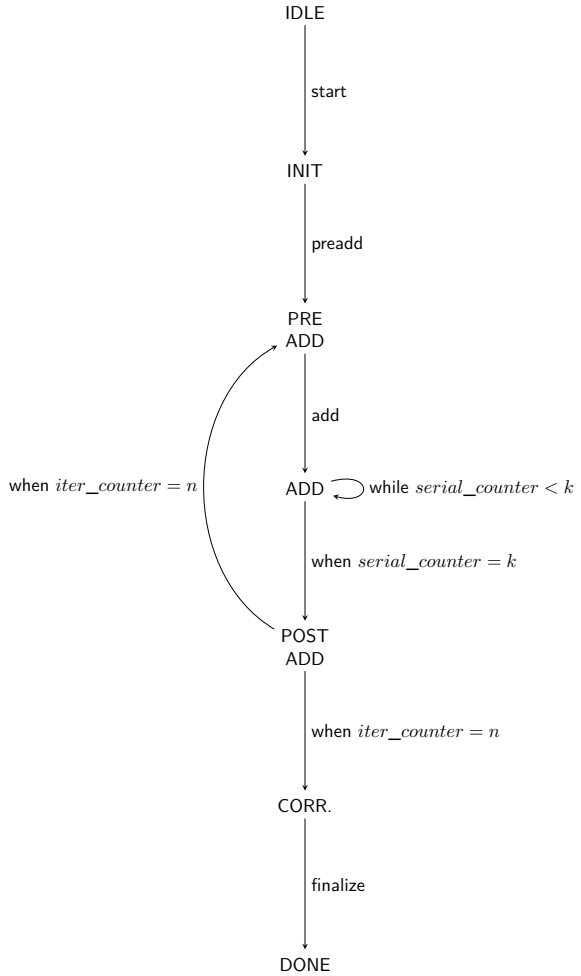


Non-restoring division circuit

Algorithm Finite State Machine



FSM baseline Non-restoring

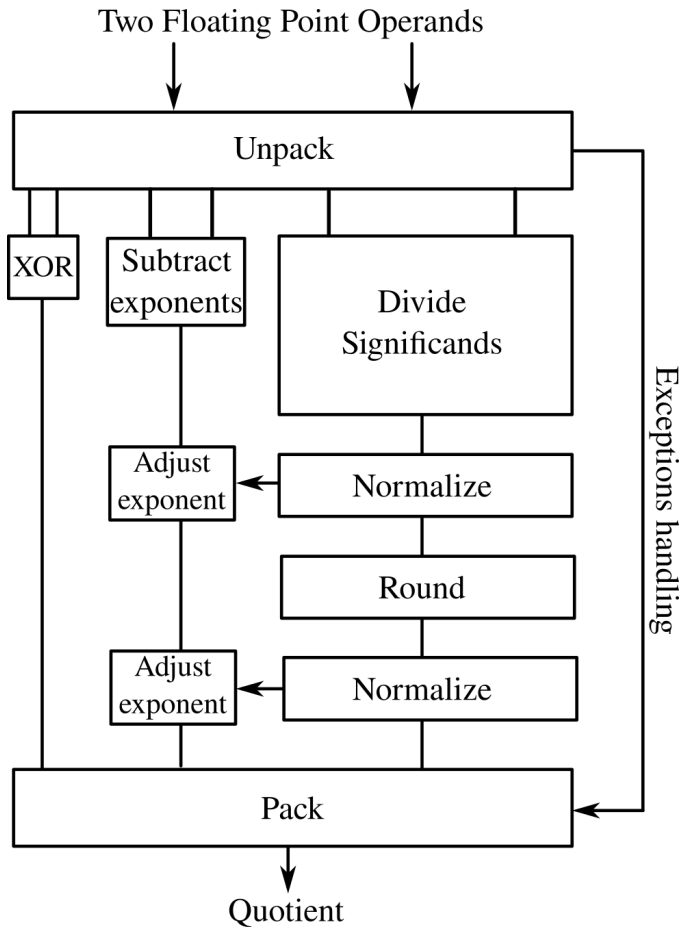


FSM modified Non-restoring

Floating-Point Integration

Three cumulative contributions:

- Dual branch baseline
- Slower Slow Path **SSP**
- Slower Fast Path **SFP**
- Slower All Path **SAP**

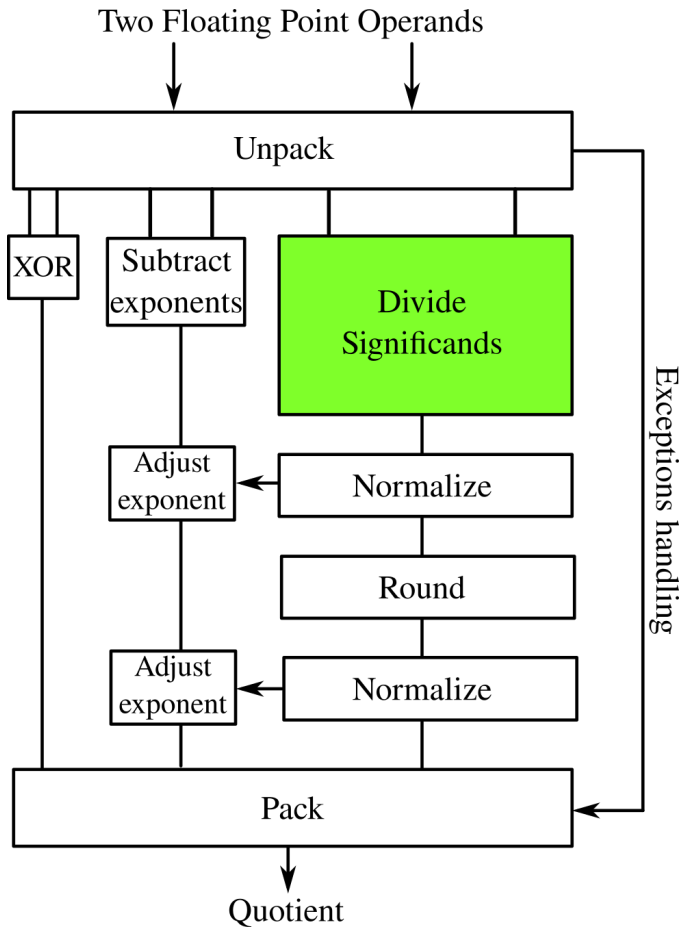


Baseline

Floating-Point Integration

Three cumulative contributions:

- Dual branch baseline
- Slower Slow Path **SSP**
- Slower Fast Path **SFP**
- Slower All Path **SAP**

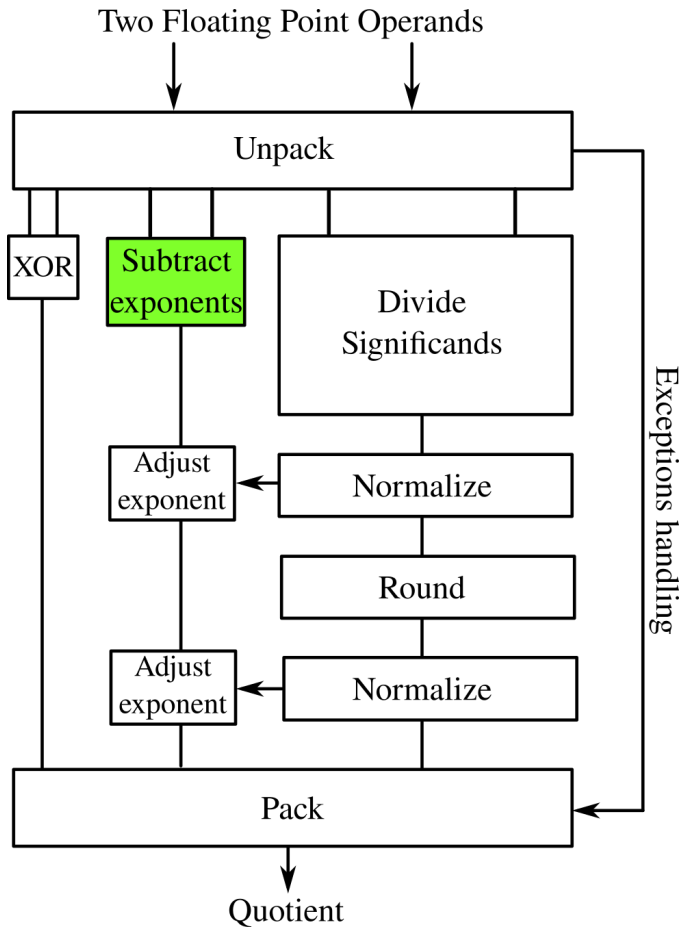


SSP

Floating-Point Integration

Three cumulative contributions:

- Dual branch baseline
- Slower Slow Path **SSP**
- Slower Fast Path **SFP**
- Slower All Path **SAP**

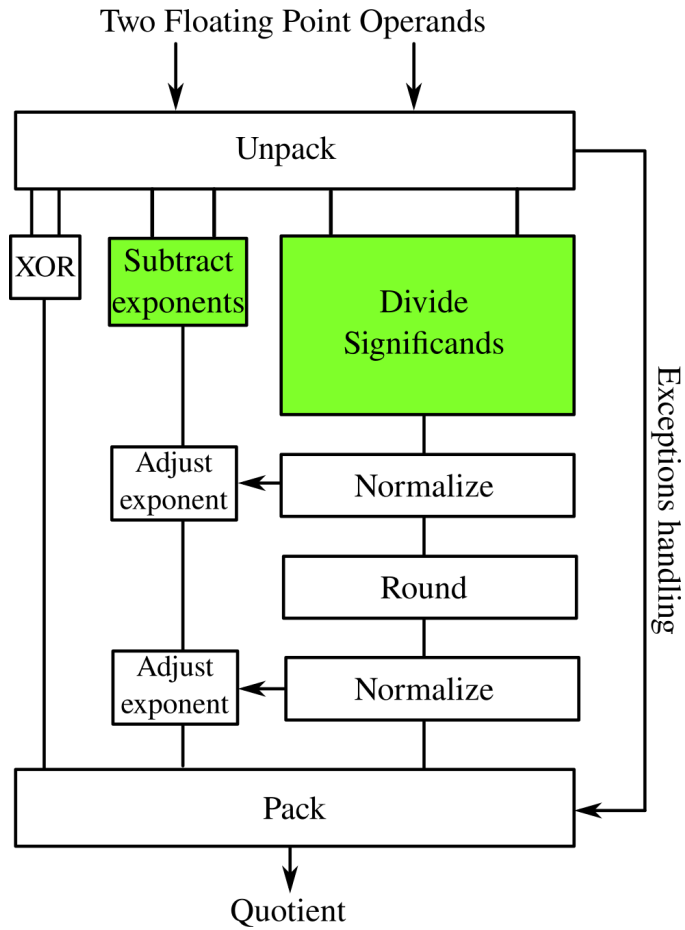


SFP

Floating-Point Integration

Three cumulative contributions:

- Dual branch baseline
- Slower Slow Path **SSP**
- Slower Fast Path **SFP**
- Slower All Path **SAP**



SAP

DSE Size Consideration

- (n_e, n_m) baseline adder sizes
- (k_e, k_m) serial adder sizes
- $S_{\mathsf{SSP}}, S_{\mathsf{SFP}}$, and S_{SAP} the experiment sets
- $|S_{\mathsf{SSP}}|, |S_{\mathsf{SFP}}|$, and $|S_{\mathsf{SAP}}|$, their cardinalities.

$$|S_{\mathsf{SAP}}| = n_m \cdot \left(n_e + 1 - \min_{k_e \in [1, n_e]} \left\{ k_e \geq \left\lceil \frac{n_e}{n_m \cdot \left\lceil \frac{n_m}{k_m} \right\rceil} \right\rceil \right\} \right)$$

Example: IEEE754 single-precision

One entry point from Arithmetic dimension: the argmin cancels out and $|S_{\mathsf{SAP}}| = n_m \cdot n_e = 8 \times 23 = 184$.

DSE Size Consideration

- (n_e, n_m) baseline adder sizes
- (k_e, k_m) serial adder sizes
- $S_{\mathsf{SSP}}, S_{\mathsf{SFP}}$, and S_{SAP} the experiment sets
- $|S_{\mathsf{SSP}}|, |S_{\mathsf{SFP}}|$, and $|S_{\mathsf{SAP}}|$, their cardinalities.

$$|S_{\mathsf{SAP}}| = n_m \cdot \left(n_e + 1 - \min_{k_e \in [1, n_e]} \left\{ k_e \geq \left\lceil \frac{n_e}{n_m \cdot \left\lceil \frac{n_m}{k_m} \right\rceil} \right\rceil \right\} \right)$$

Example: IEEE754 single-precision

One entry point from Arithmetic dimension: the argmin cancels out and $|S_{\mathsf{SAP}}| = n_m \cdot n_e = 8 \times 23 = 184$.

DSE Size Consideration

11 Float Formats

IEEE754QP with parameters $(n_e = 15, n_m = 112)$, **IEEE754DP** with parameters $(n_e = 11, n_m = 52)$, **IEEE754SP** with parameters $(n_e = 8, n_m = 23)$, **IEEE754HP** with parameters $(n_e = 5, n_m = 10)$, **BrainFloat16** with $(n_e = 8, n_m = 7)$, **Posit**$\langle \mathbf{64}, \mathbf{2} \rangle$ with $(n_e = 9, n_m = 59)$, **Posit**$\langle \mathbf{32}, \mathbf{2} \rangle$ with $(n_e = 8, n_m = 27)$, **Posit**$\langle \mathbf{16}, \mathbf{2} \rangle$ with $(n_e = 7, n_m = 9)$, **Posit**$\langle \mathbf{8}, \mathbf{2} \rangle$ with $(n_e = 6, n_m = 3)$, **E5M2** with $(n_e = 5, n_m = 2)$, and **E4M3** with $(n_e = 4, n_m = 3)$. Totalling in **11** evaluated floats.

Functional Specifications

A total of **227** SSP designs.

Performance Specifications

Combined with 5 PDKs (ranging from 180 nm to 7 nm), culminating in **1135** chips to evaluate.

DSE Size Consideration

11 Float Formats

IEEE754QP with parameters $(n_e = 15, n_m = 112)$, **IEEE754DP** with parameters $(n_e = 11, n_m = 52)$, **IEEE754SP** with parameters $(n_e = 8, n_m = 23)$, **IEEE754HP** with parameters $(n_e = 5, n_m = 10)$, **BrainFloat16** with $(n_e = 8, n_m = 7)$, **Posit**$\langle \mathbf{64}, \mathbf{2} \rangle$ with $(n_e = 9, n_m = 59)$, **Posit**$\langle \mathbf{32}, \mathbf{2} \rangle$ with $(n_e = 8, n_m = 27)$, **Posit**$\langle \mathbf{16}, \mathbf{2} \rangle$ with $(n_e = 7, n_m = 9)$, **Posit**$\langle \mathbf{8}, \mathbf{2} \rangle$ with $(n_e = 6, n_m = 3)$, **E5M2** with $(n_e = 5, n_m = 2)$, and **E4M3** with $(n_e = 4, n_m = 3)$. Totalling in **11** evaluated floats.

Functional Specifications

A total of **227** SSP designs.

Performance Specifications

Combined with 5 PDKs (ranging from 180 nm to 7 nm), culminating in **1135** chips to evaluate.

DSE Size Consideration

11 Float Formats

IEEE754QP with parameters $(n_e = 15, n_m = 112)$, **IEEE754DP** with parameters $(n_e = 11, n_m = 52)$, **IEEE754SP** with parameters $(n_e = 8, n_m = 23)$, **IEEE754HP** with parameters $(n_e = 5, n_m = 10)$, **BrainFloat16** with $(n_e = 8, n_m = 7)$, **Posit**$\langle \mathbf{64}, \mathbf{2} \rangle$ with $(n_e = 9, n_m = 59)$, **Posit**$\langle \mathbf{32}, \mathbf{2} \rangle$ with $(n_e = 8, n_m = 27)$, **Posit**$\langle \mathbf{16}, \mathbf{2} \rangle$ with $(n_e = 7, n_m = 9)$, **Posit**$\langle \mathbf{8}, \mathbf{2} \rangle$ with $(n_e = 6, n_m = 3)$, **E5M2** with $(n_e = 5, n_m = 2)$, and **E4M3** with $(n_e = 4, n_m = 3)$. Totalling in **11** evaluated floats.

Functional Specifications

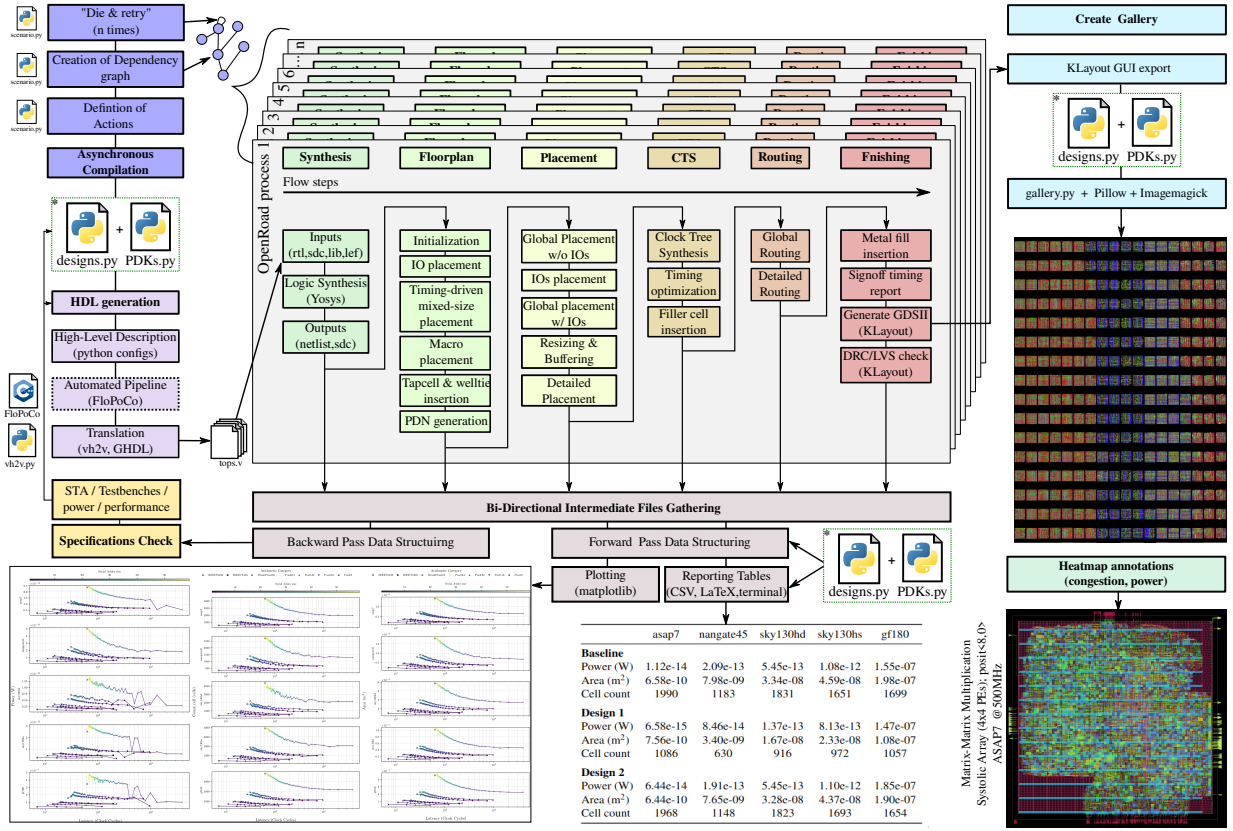
A total of **227** SSP designs.

Performance Specifications

Combined with 5 PDKs (ranging from 180 nm to 7 nm), culminating in **1135** chips to evaluate.

SUF: SUperset Framework, a grafted approach to ASIC

- High-level language to GDS
- Asynchronous and parralel
- 1135 chips in < 2 hours

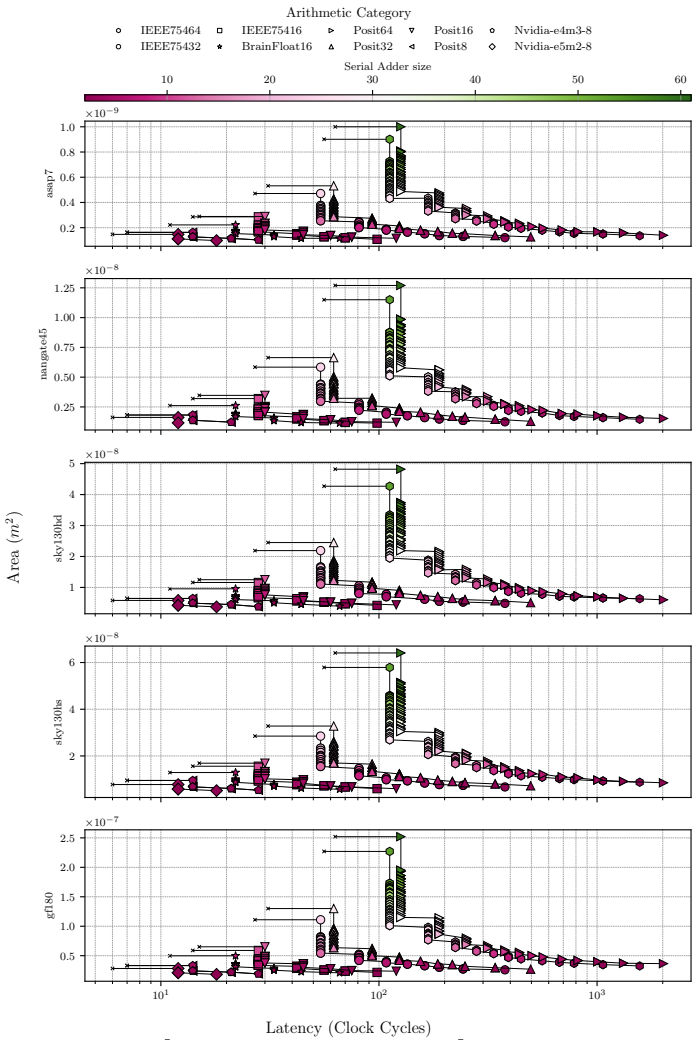


All features of SUF

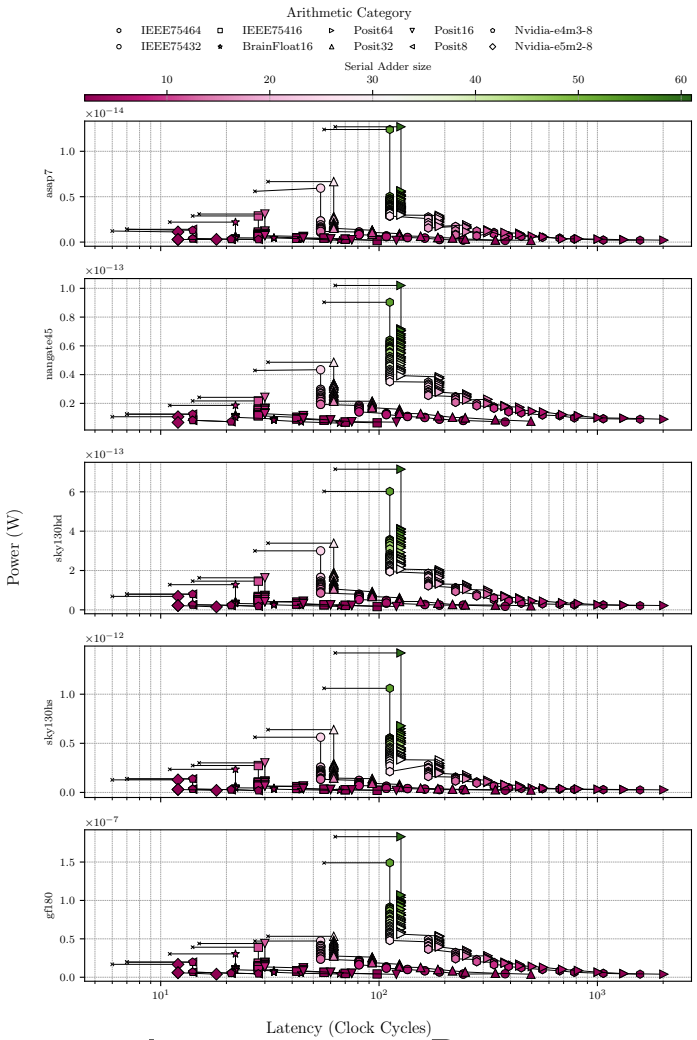
Current Progress

Floating-Point Divisions

Results: SSP Latency VS. Power and Area

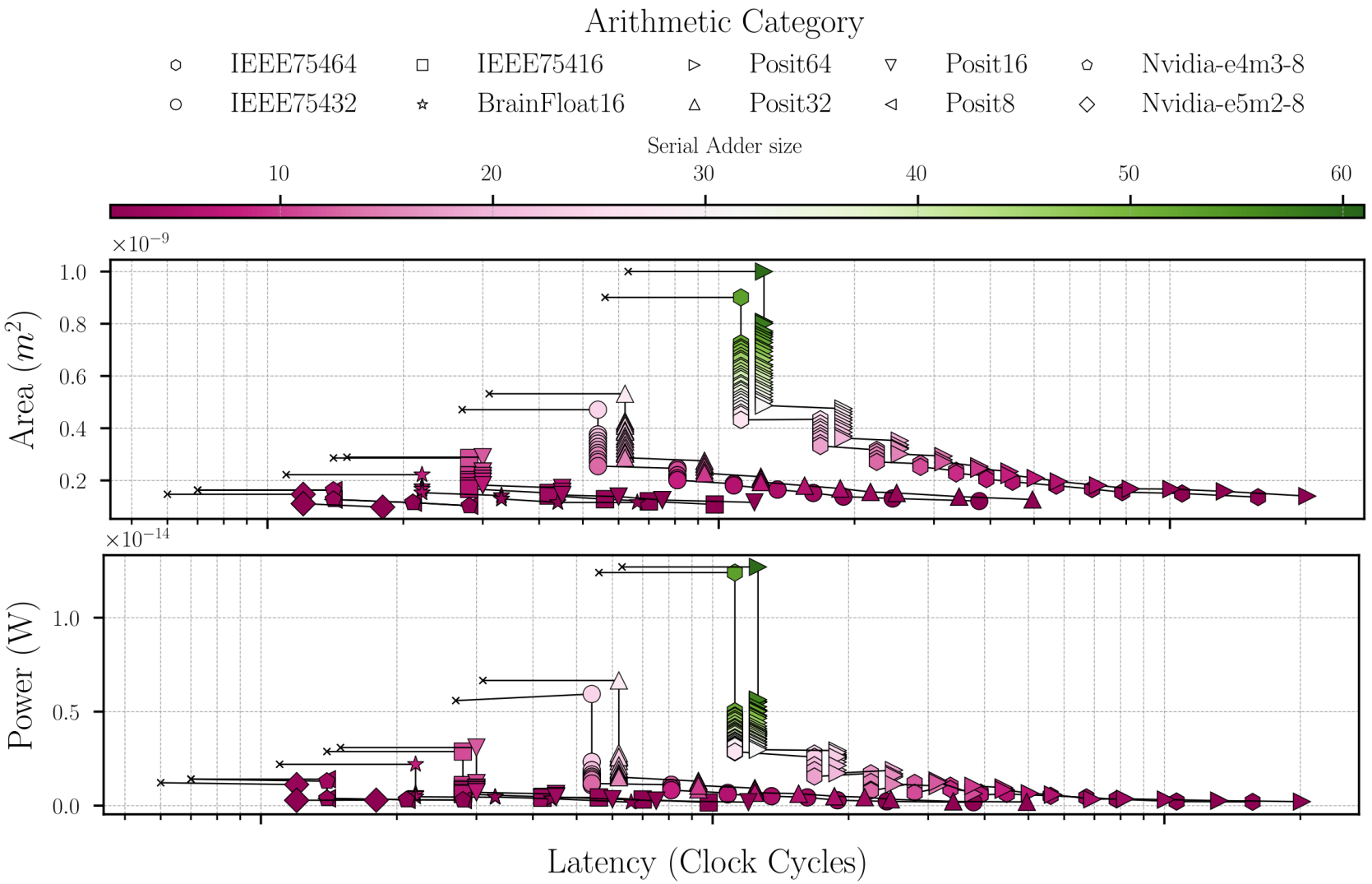


Latency vs. Area



Latency vs. Power

Results: Latency vs. metrics zoom on ASAP7

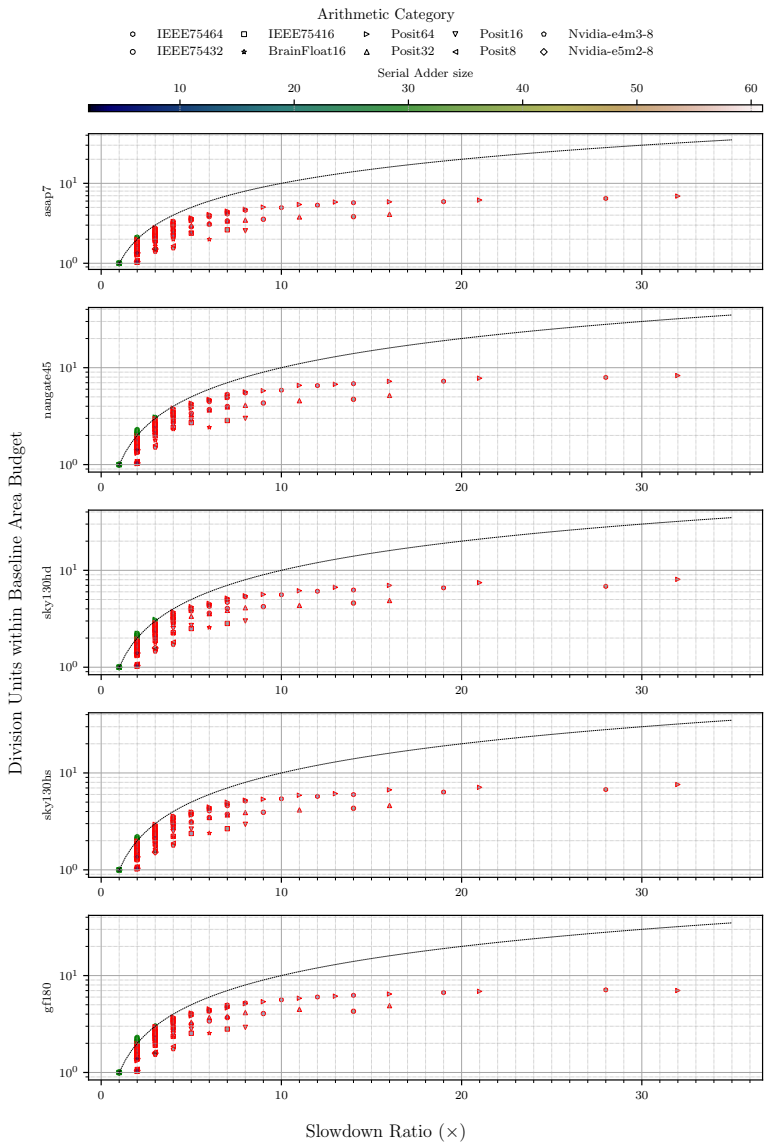


ASAP7 Latency vs. Area and Power.

Results: SAP Area Efficiency

Experiment:

- Latency VS. Parallelism
- **Performance line**
- Baseline: Non-Serial Non-Restoring
- E.g. 2× slower ⇒ 2× smaller ⇒ 2× more units



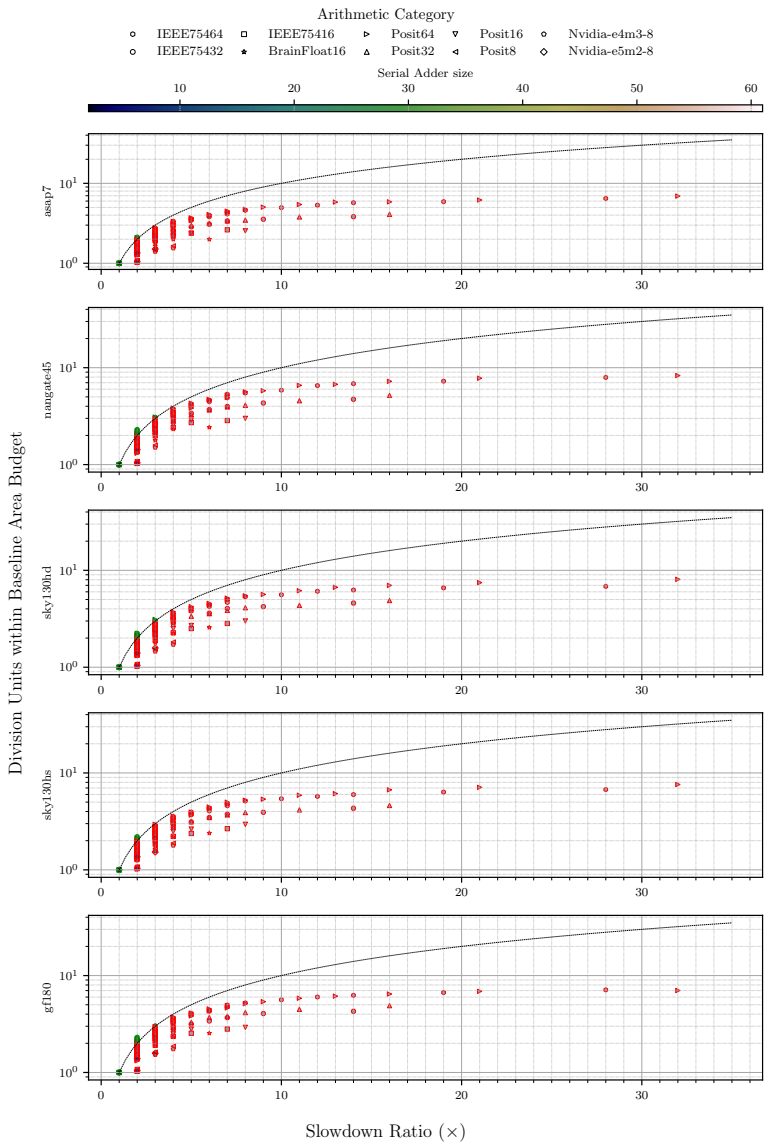
Results: SAP Area Efficiency

Experiment:

- Latency VS. Parallelism
- **Performance line**
- Baseline: Non-Serial Non-Restoring
- E.g. 2× slower ⇒ 2× smaller ⇒ 2× more units

Results:

- **Improved Area Efficiency:** 9.40%.
- **Area gain at Slowdown 2:** $2.32\times$.
- **Area gain at Slowdown 3:** $3.12\times$.
- **No Gains from $4\times$ Slowdown.**

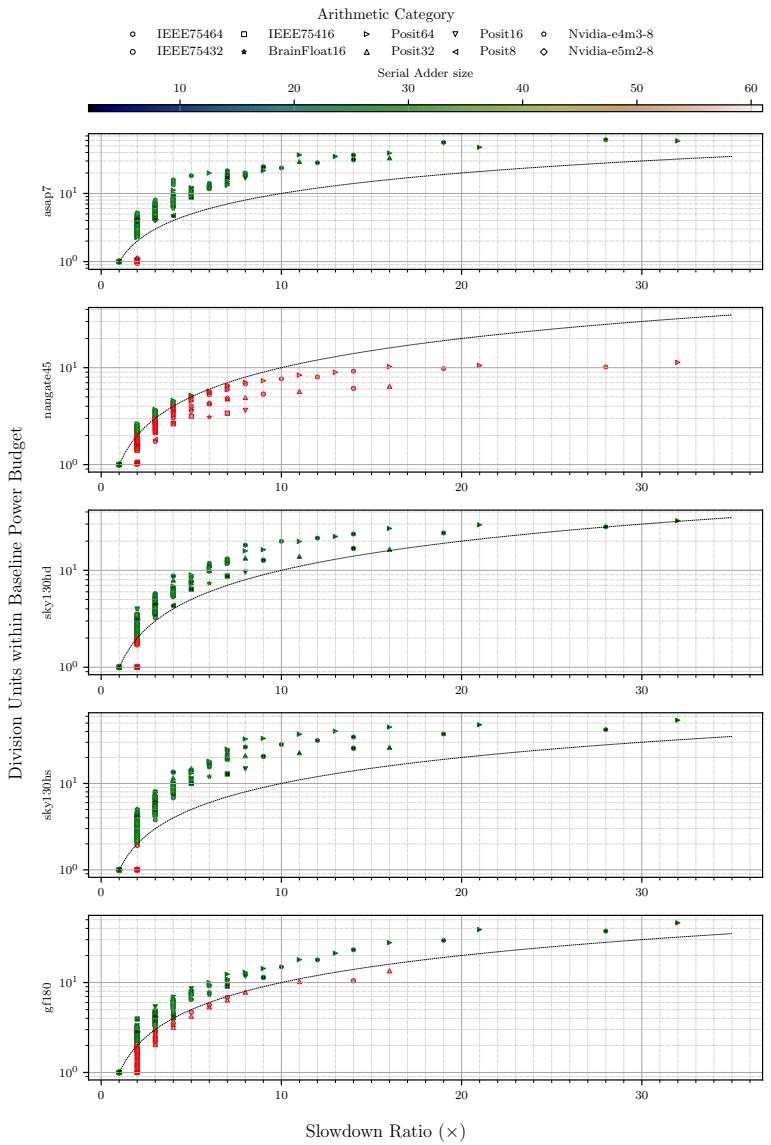


Results: SAP Power Efficiency

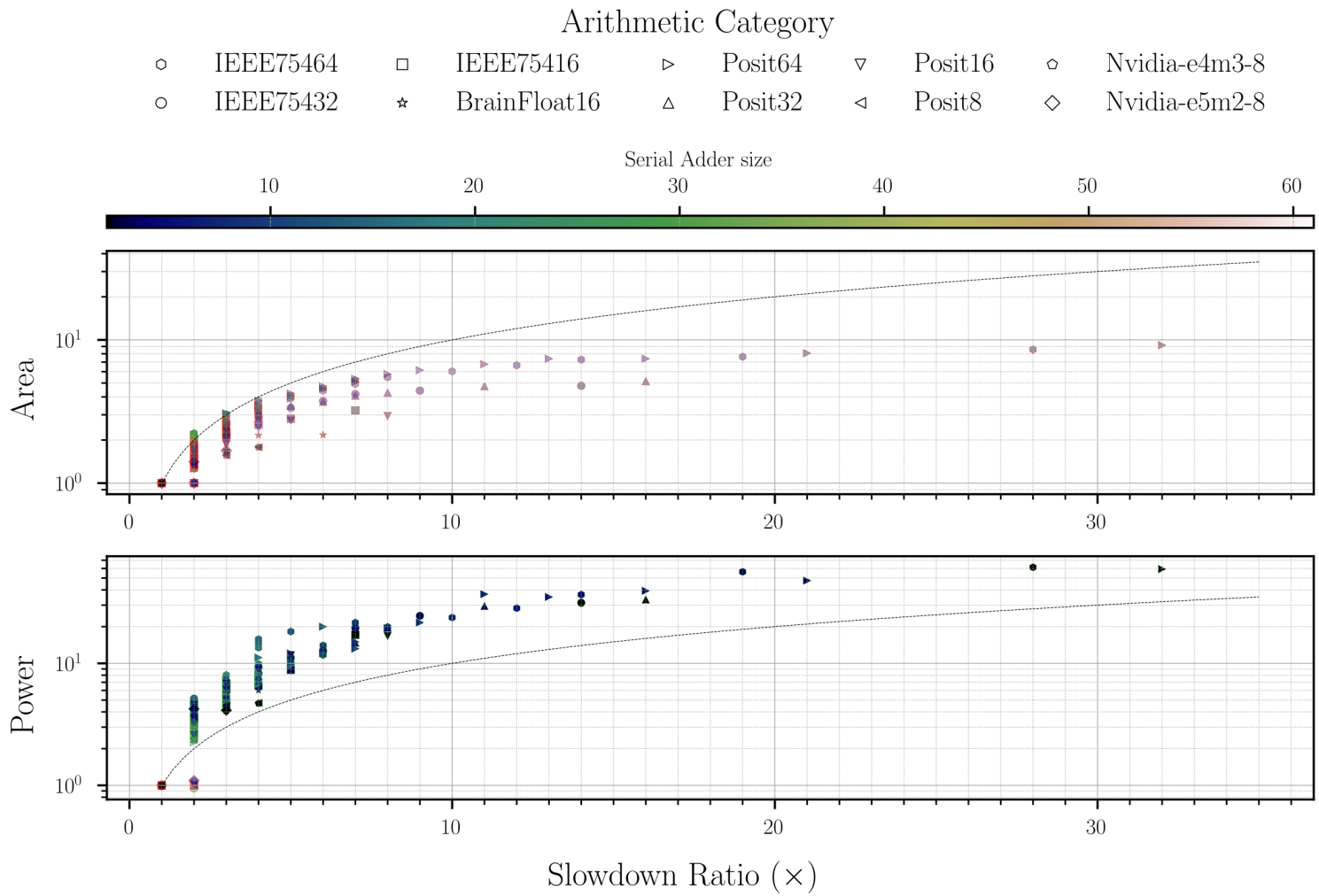
Experiment: baseline budget is **power**.

Results:

- **Improved Power Efficiency:** 68.66%.
- **Best Power gain:** $61.28\times$.
- **Power gain at Slowdown 2:** $5.14\times$.
- **Power gain at Slowdown 3:** $8.03\times$.



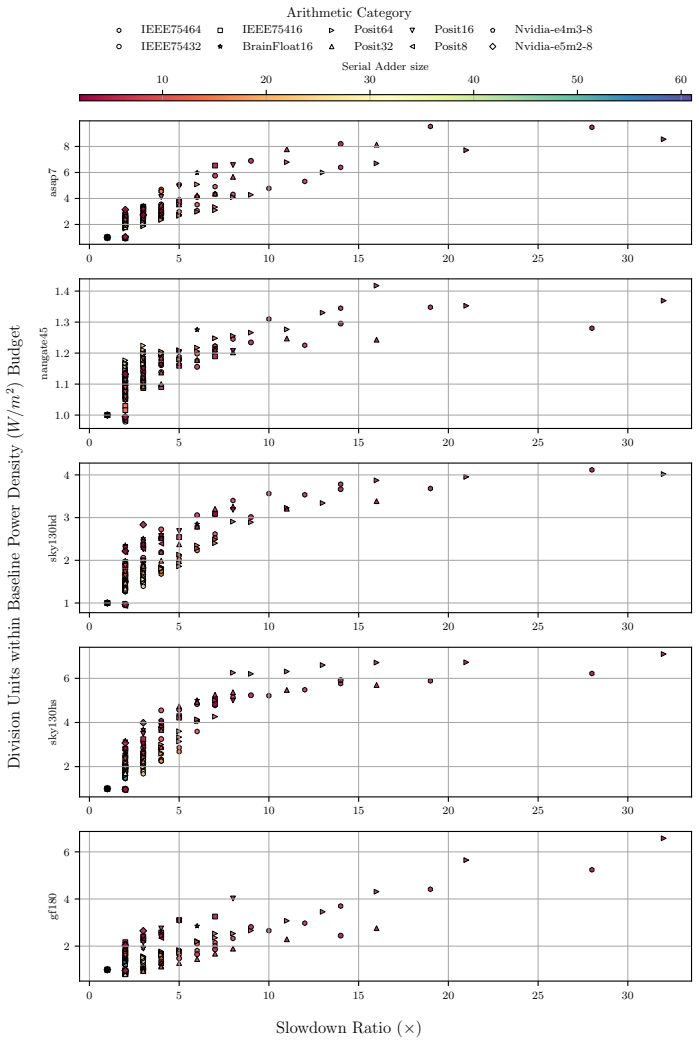
Results: Area and Power zooms on ASAP7



How many more units in ASAP7 for Area and Power budgets.

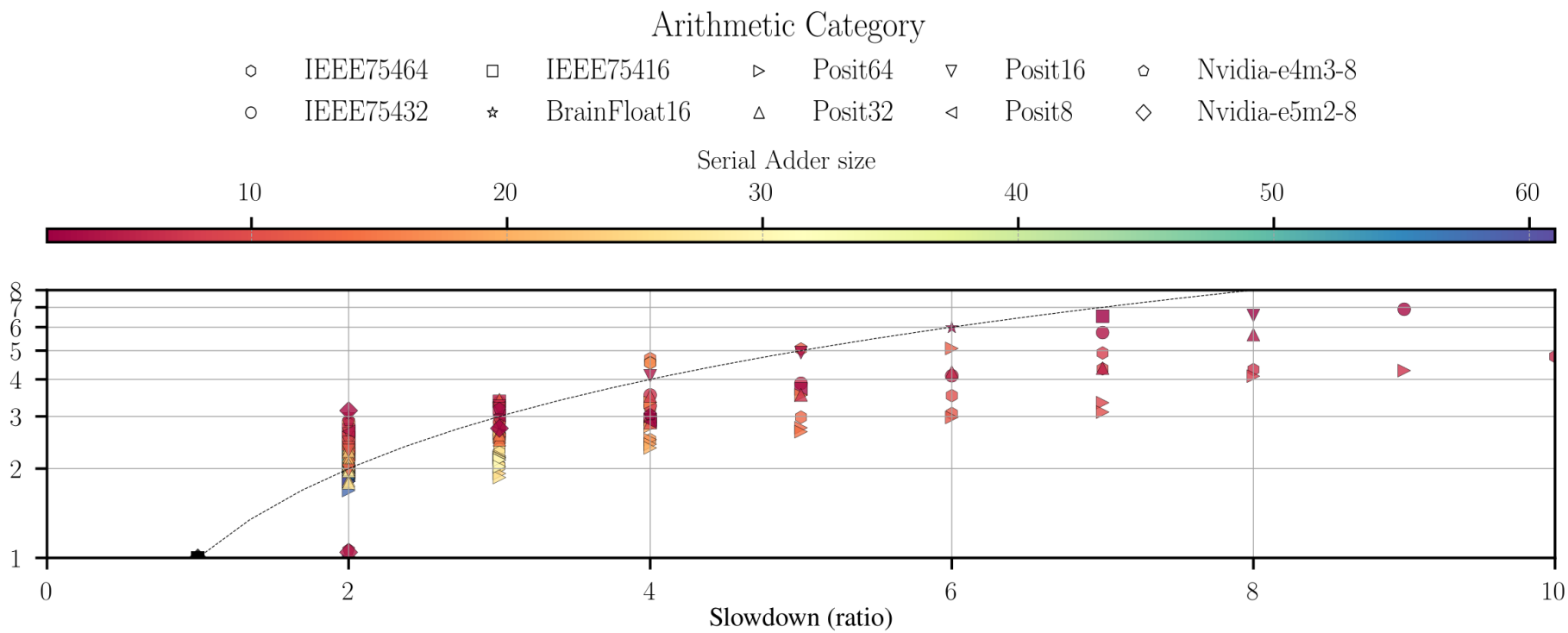
Results: Combining Power and Area

- **Power Density:** W/m^2.
- **Efficent Thermal Management.**
- **Ensure cooling of many smaller units.**



SAP Power Density Efficiency

Results: Power density zoom on ASAP7



How many more units in ASAP7 for Power Density budgets.

Current Progress

Floating-Point Divisions

Summary of Insights and Key Contributions

1135 chips

Our SUF ASIC flow provides insights on **1135** division algorithms.

Vector Processing

Exhibition and enchancement of **Data-Level-Parallelism** with more slower and smaller units.

Improved Efficiencies

Our units show **improved area, power, and power density** gains.

Power Wall and Dark Silicon

Matching the paradigm of vector processing up to the execution unit **answers the dark silicon concern**.

Summary of Insights and Key Contributions

1135 chips

Our SUF ASIC flow provides insights on **1135** division algorithms.

Vector Processing

Exhibition and enchancement of **Data-Level-Parallelism** with more slower and smaller units.

Improved Efficiencies

Our units show **improved area, power, and power density** gains.

Power Wall and Dark Silicon

Matching the paradigm of vector processing up to the execution unit **answers the dark silicon concern**.

Summary of Insights and Key Contributions

1135 chips

Our SUF ASIC flow provides insights on **1135** division algorithms.

Vector Processing

Exhibition and enchancement of **Data-Level-Parallelism** with more slower and smaller units.

Improved Efficiencies

Our units show **improved area, power, and power density** gains.

Power Wall and Dark Silicon

Matching the paradigm of vector processing up to the execution unit **answers the dark silicon concern**.

Summary of Insights and Key Contributions

1135 chips

Our SUF ASIC flow provides insights on **1135** division algorithms.

Vector Processing

Exhibition and enchancement of **Data-Level-Parallelism** with more slower and smaller units.

Improved Efficiencies

Our units show **improved area, power, and power density** gains.

Power Wall and Dark Silicon

Matching the paradigm of vector processing up to the execution unit **answers the dark silicon concern**.

Agenda

Introduction

Posit AI acceleration

Generator of Systolic Arrays

Numerically-tailored BLAS

Floating-Point Divisions

Closing

Conclusions

First Contribution

Provided early insight in an **HPC environment** of an emerging floating-point format: **Posits**.

Second Contribution

Ensured **fairness** between formats comparison and provided a **generator of accelerated GEMM** kernels focused on **arithmetic**.

Third Contribution

Connected **silicon layout** to **HPC software**. Systematic enchancement **accuracy and energy** consumption for **distinct HPC codes**.

Last Contribution

Revisited **Circuitry** for floating-point **division** motivated by modern opportunities: **Vector Processing** and **Open Silicon**.

Conclusions

First Contribution

Provided early insight in an **HPC environment** of an emerging floating-point format: **Posits**.

Second Contribution

Ensured **fairness** between formats comparison and provided a **generator of accelerated GEMM** kernels focused on **arithmetic**.

Third Contribution

Connected **silicon layout** to **HPC software**. Systematic enchancement **accuracy and energy** consumption for **distinct HPC codes**.

Last Contribution

Revisited **Circuitry** for floating-point **division** motivated by modern opportunities: **Vector Processing** and **Open Silicon**.

Conclusions

First Contribution

Provided early insight in an **HPC environment** of an emerging floating-point format: **Posits**.

Second Contribution

Ensured **fairness** between formats comparison and provided a **generator of accelerated GEMM** kernels focused on **arithmetic**.

Third Contribution

Connected **silicon layout** to **HPC software**. Systematic enchancement **accuracy and energy** consumption for **distinct HPC codes**.

Last Contribution

Revisited **Circuitry** for floating-point **division** motivated by modern opportunities: **Vector Processing** and **Open Silicon**.

Conclusions

First Contribution

Provided early insight in an **HPC environment** of an emerging floating-point format: **Posits**.

Second Contribution

Ensured **fairness** between formats comparison and provided a **generator of accelerated GEMM** kernels focused on **arithmetic**.

Third Contribution

Connected **silicon layout** to **HPC software**. Systematic enchancement **accuracy and energy** consumption for **distinct HPC codes**.

Last Contribution

Revisited **Circuitry** for floating-point **division** motivated by modern opportunities: **Vector Processing** and **Open Silicon**.

Dissemination: Publications

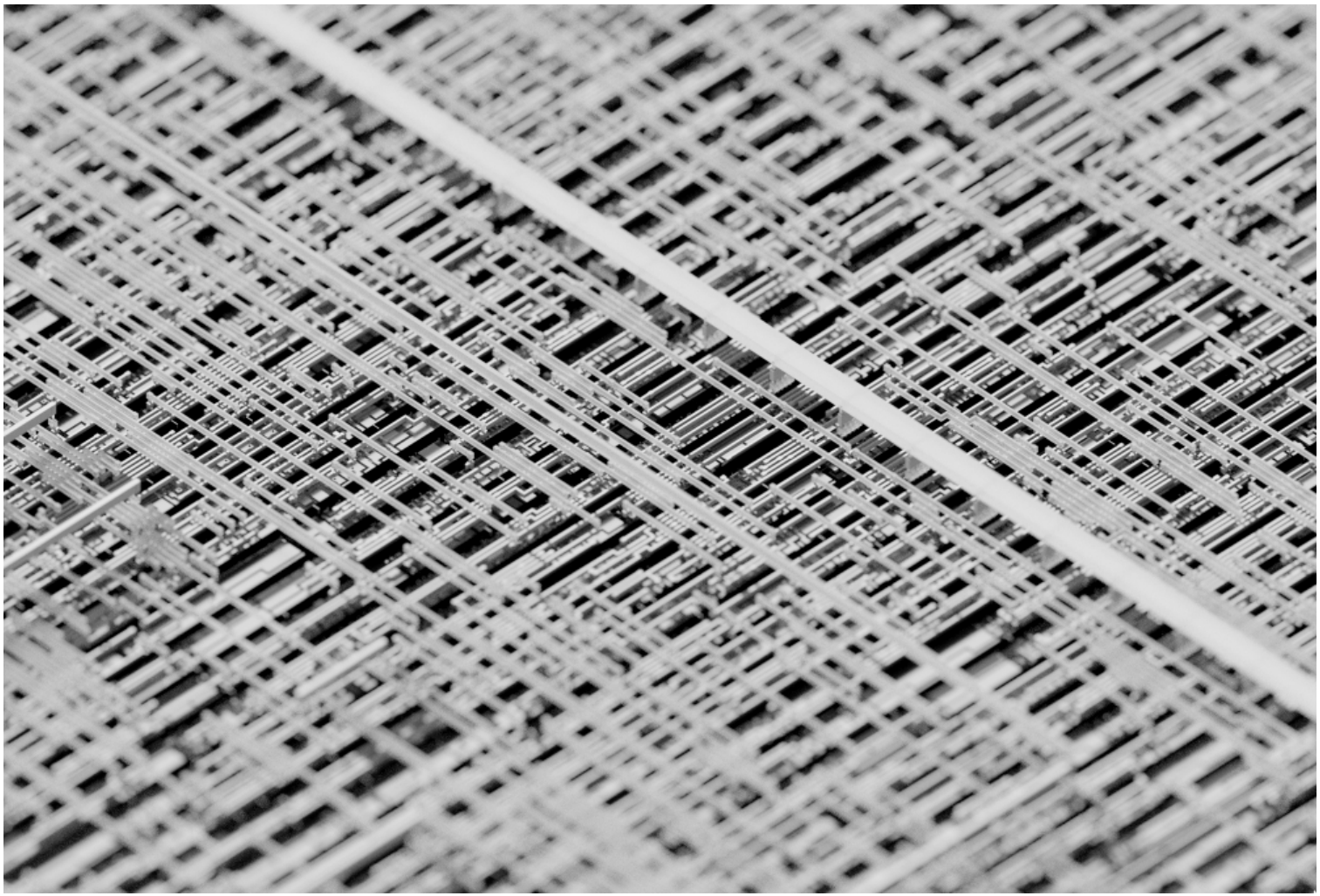
- **L. Ledoux** and M. Casas, "A Generator of Numerically-Tailored and High-Throughput Accelerators for Batched GEMMs," in 2022 IEEE 30th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM), May 2022, pp. 1–10. *doi: 10.1109/FCCM53951.2022.9786164*
- **L. Ledoux** and M. Casas, "An Open-Source Framework for Efficient Numerically-Tailored Computations," in 2023 33rd International Conference on Field-Programmable Logic and Applications (FPL), Sep. 2023, pp. 19–26. *doi: 10.1109/FPL60245.2023.00011*
- **L. Ledoux** and M. Casas, "The Grafted Superset Approach: Bridging Python to Silicon with Asynchronous Compilation and Beyond" in 2024 4th Workshop on Open-Source Design Automation (OSDA), hosted at DATE, March 25, 2024, at Palacio De Congresos Valencia, Spain. *Available online soon.*
- **L. Ledoux** and M. Casas, "LLMMMM: Large Language Models Matrix-Matrix Multiplications Characterization on Open Silicon" in 2024 11th BSCSymposium, May 2024, *Available online soon.*
- **L. Ledoux** and M. Casas, "Open-Source GEMM Hardware Kernels Generator: Toward Numerically-Tailored Computations" in 2023 10th BSCSymposium, May 2023, *Available: https://arxiv.org/abs/2305.18328*
- **L. Ledoux** and M. Casas, "Accelerating DL inference with (Open)CAPI and posit numbers," in OpenPOWER summit 2019, Lyon, France: linux foundation, Oct. 2019. *Available: https://hal.science/hal-04094850*

Open Source Thesis: Reproducibility and Collaborations

- **OSFNTC:** Implementation of the Open-Source Framework for Numerically-Tailored Computations. *Available:* *https://github.com/Bynaryman/OSFNTC*
- **Teras and Wrapped_teras:** Code and GDS files for the Systolic Array with posits<8,0> on the SkyWater shuttle. *Available:* *https://github.com/Bynaryman/teras*, *https://github.com/Bynaryman/wrapped_teras*.
- **POF (Posit Operator Framework):** Suite for developing and testing posit-based arithmetic operations. *Available:* *https://github.com/Bynaryman/POF*
- **SUF (Superset Framework):** Enhances ASIC flow by adding asynchronous parallelism. *Available:* *https://github.com/Bynaryman/SUF*
- **3lnn (Three-Layer Neural Network):** Neural network topology and pre-trained weights in posit format. *Available:* *https://github.com/Bynaryman/mnist-3lnn*
- **Flopoco / Virtex FPGA family:** Pipeline tool for arithmetic operators used in FPGA design. *Commit:* *https://gitlab.com/flopoco/flopoco/-/commit/0e0db94d2c5dc1084d477b9cb9b8b34d1a23a9e1*
- **VH2V:** VHDL to Verilog translator optimized for arithmetic operations. *Available:* *https://github.com/Bynaryman/vh2v*

Is that Ada indicating the repositories' URL?

Systolic Array Tapeout



Raytracing Render of the GDS polygons.

Systolic Array Tapeout

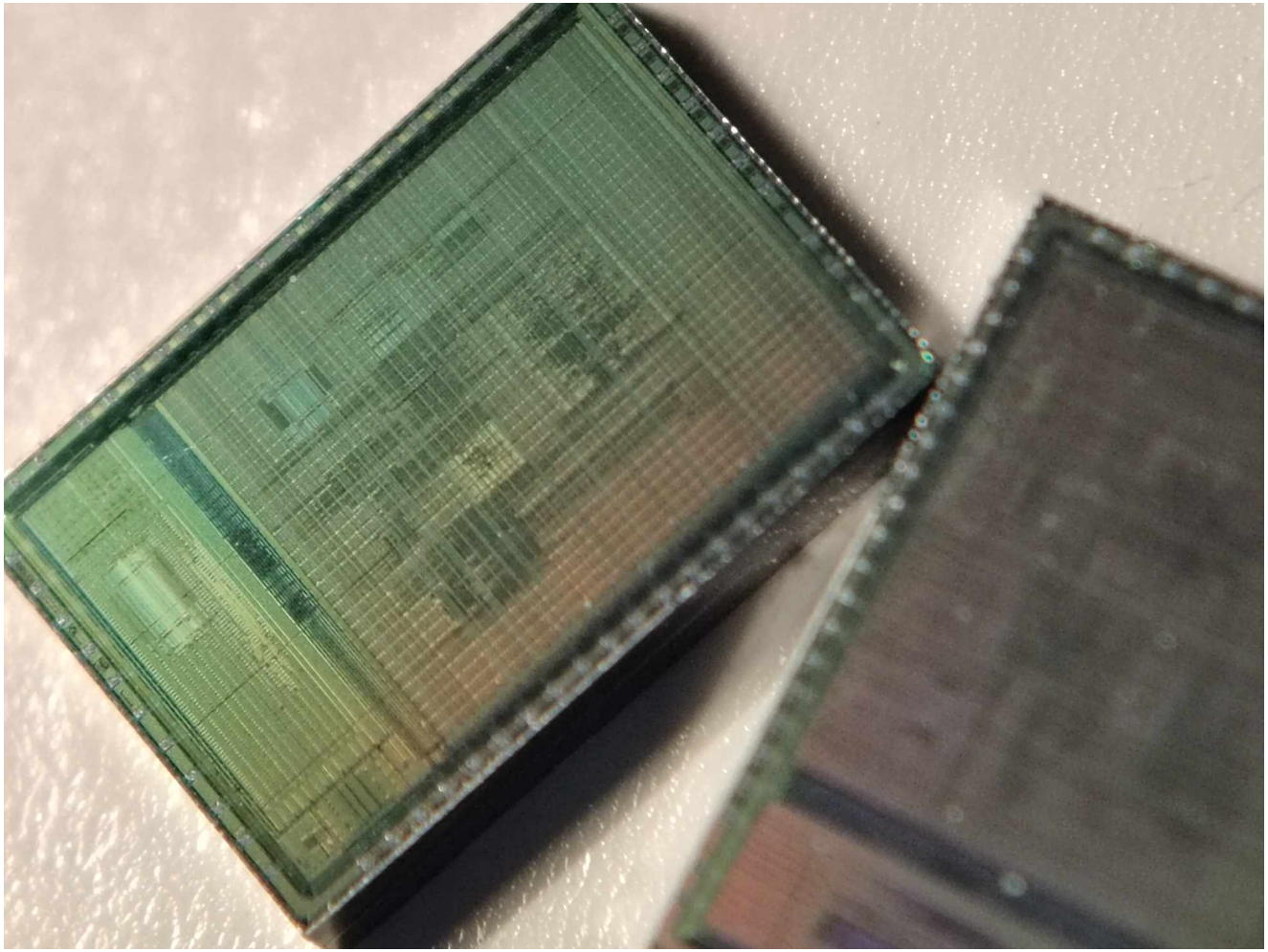
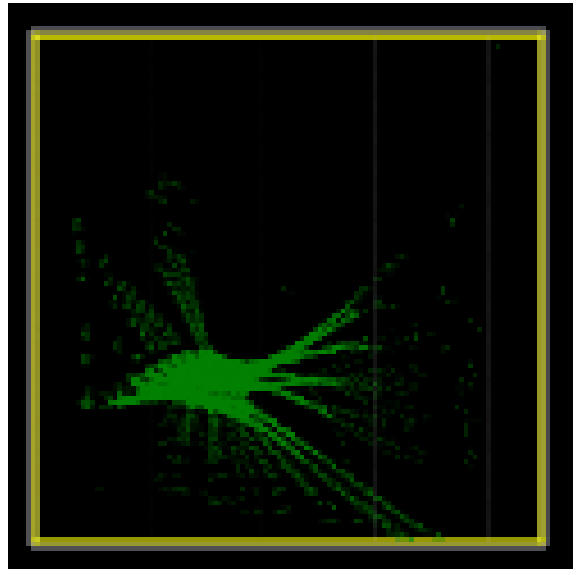


Photo taken with a lens and a smartphone.



Thank You

Absolute and Relative Errors

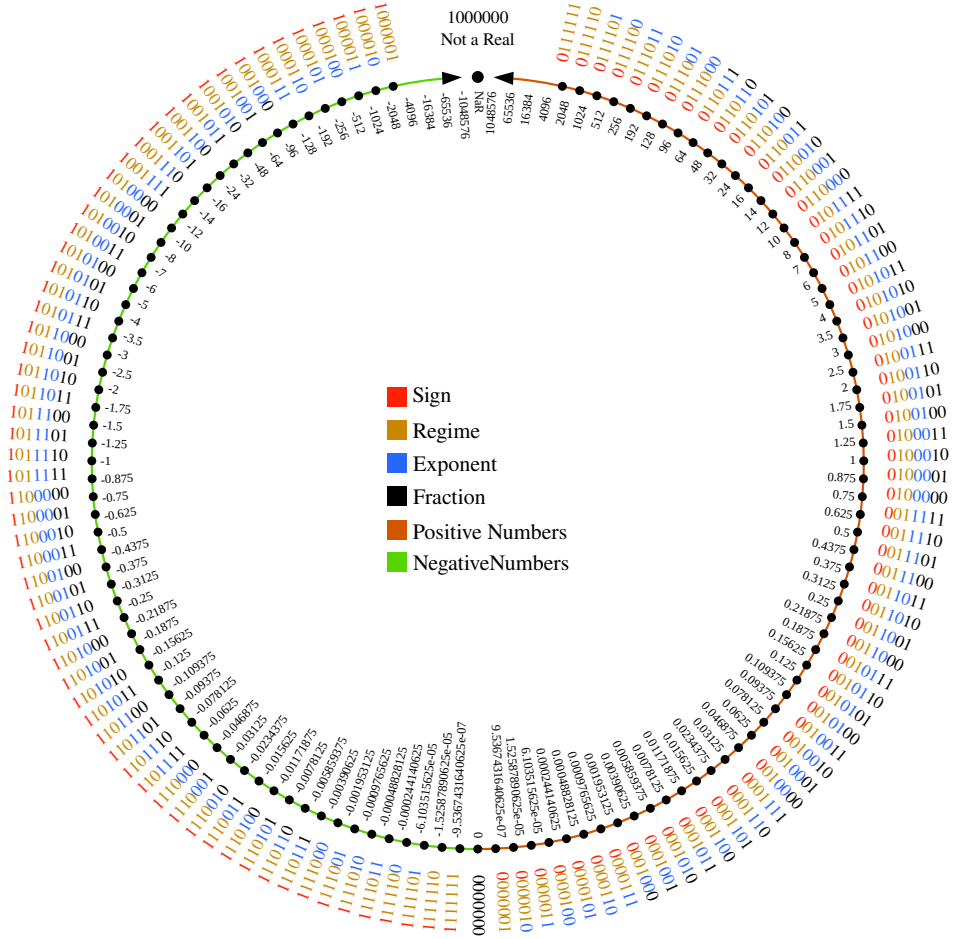
Error Analysis and Formulas

$$\text{Absolute Error} = |R_{\text{exact}} - R|$$

$$\text{Relative Error} = \frac{|R_{\text{exact}} - R|}{|R_{\text{exact}}|}$$

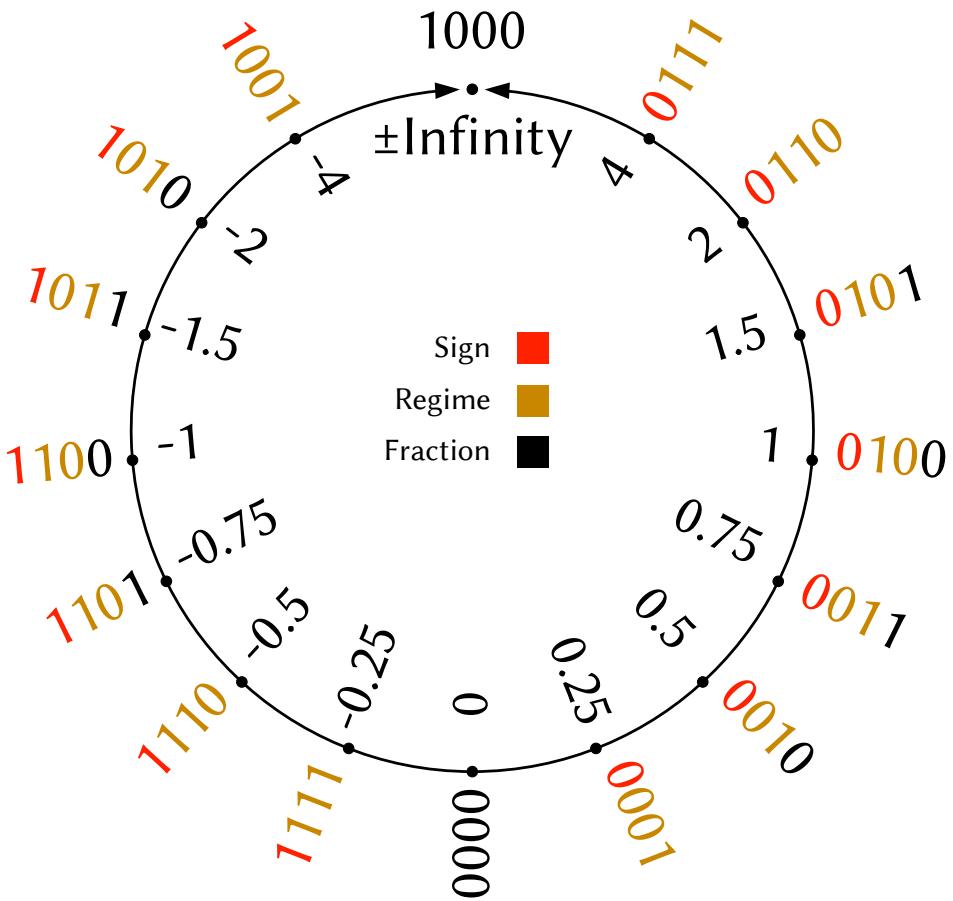
	Absolute Error	**Relative Error**
FMA	1.46044921875	0.0311 (3.11%)
Kulisch	0	0

Background: Posit representations



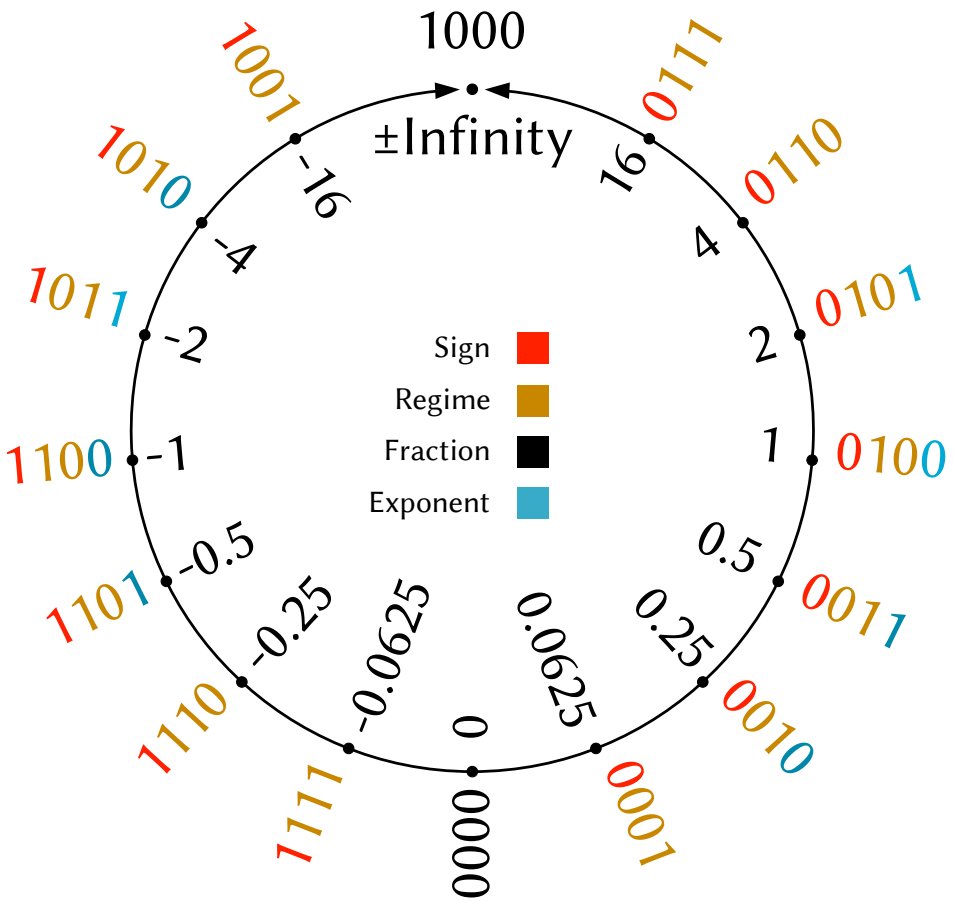
$posit\ \langle 7, 2 \rangle$.

Background: Posit representations



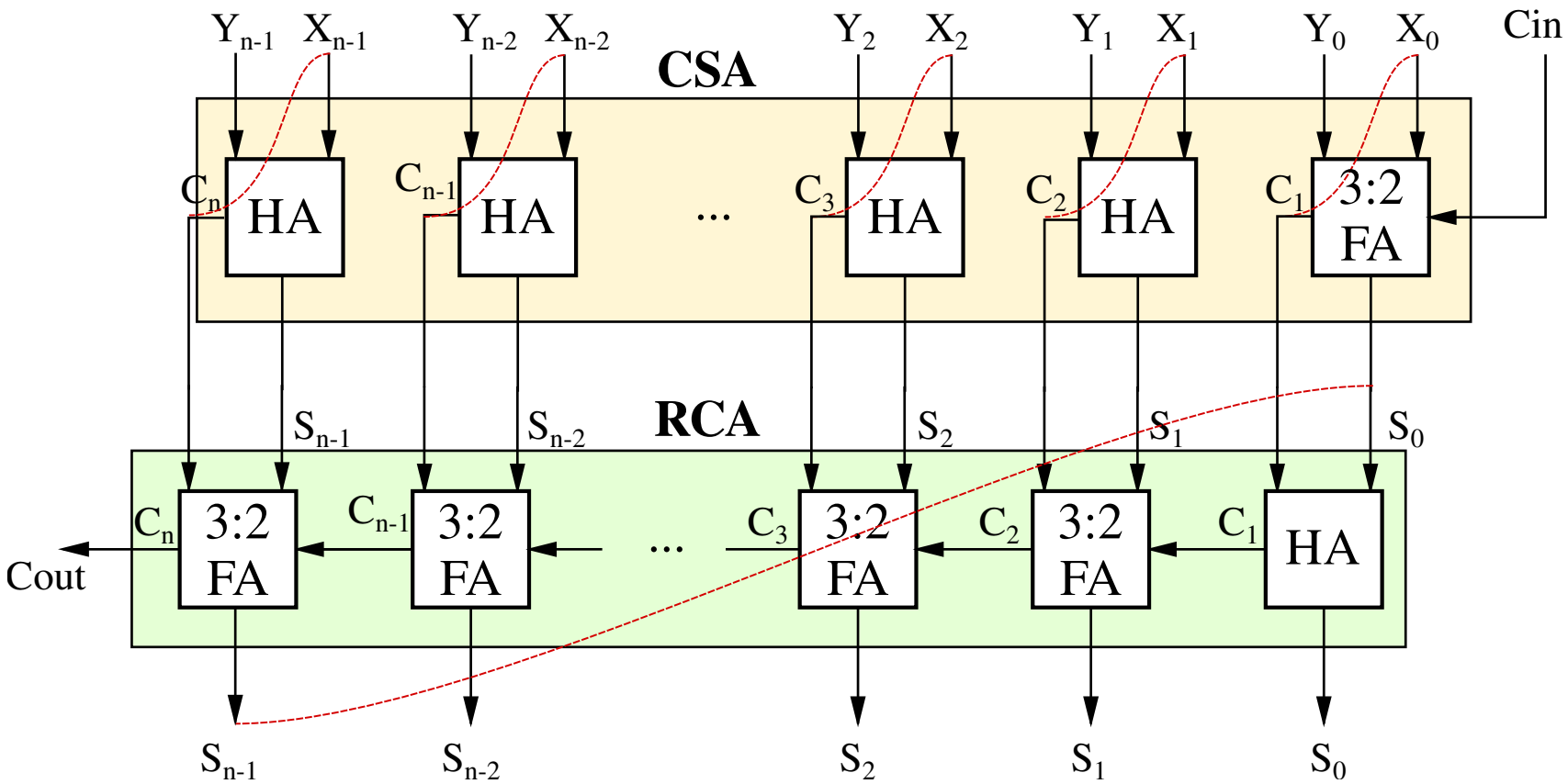
$posit\ \langle 4, 0 \rangle$.

Background: Posit representations



$posit\ \langle 4, 1 \rangle$.

CSA with RCA



CSA followed by RCA with critical path in red.

Background: Posit Decoding Example

Posit Encoding Formula

Given a posit number x, it can be decoded using the formula:

$$x = \begin{cases} 0 & \text{when } 0\ldots0 \\ \pm\infty & \text{when } 10\ldots0 \\ (-1)^s \times \text{useed}^k \times 2^e \times \left(1 + \frac{f_{m-1}\cdots f_0}{2^m}\right) & \text{otherwise} \end{cases}$$

Example: posit<5,1> Ob00101

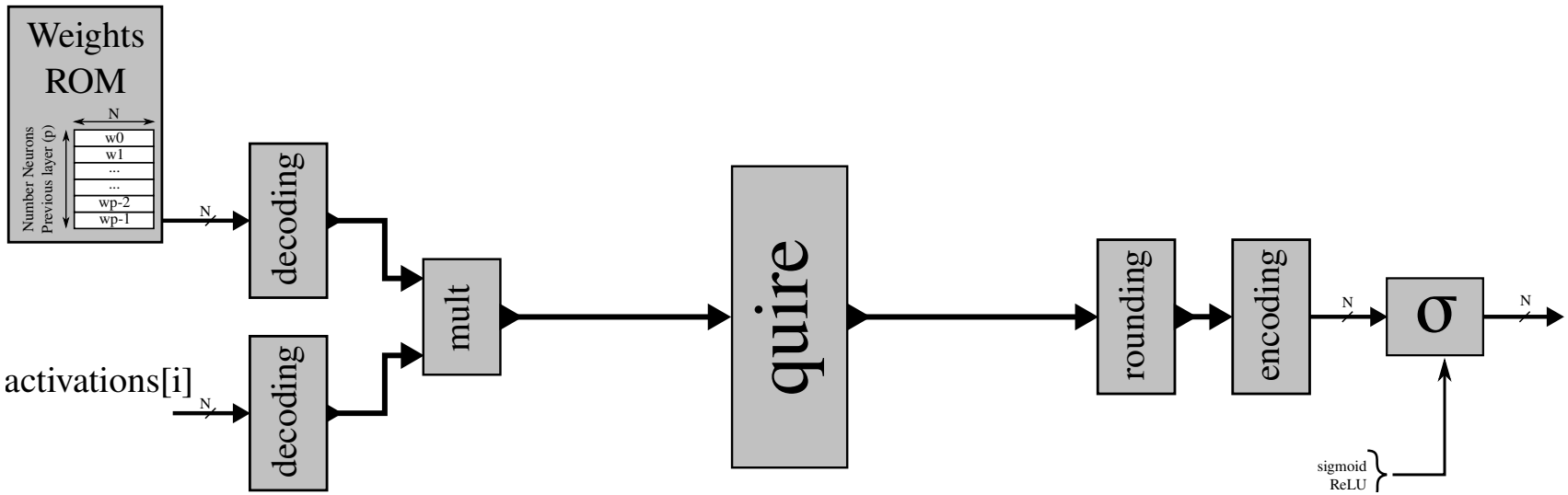
- $k = -1$ (because 1 run-bit (0) followed by the r bit (1))
- $x = (-1)^0 \times (2^{2^1})^{-1} \times 2^0 \times \left(1 + \frac{1}{2^1}\right)$
- $x = 1 \times \frac{1}{4} \times 1 \times \frac{3}{2}$
- $x = \frac{3}{8}$

Methodology Overview

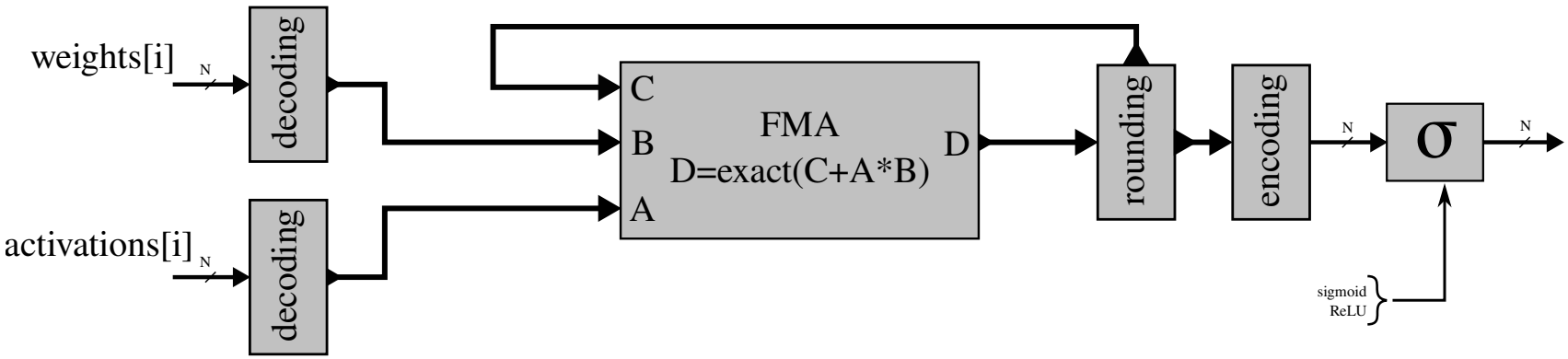
Parallel Approach

We adopt a **parallel approach**, combining a **bottom-up** methodology for the arithmetic part with a **horizontal** strategy for HPC integration.

Neuron $\langle N, es \rangle$ Variations: 3 Posit Implementations

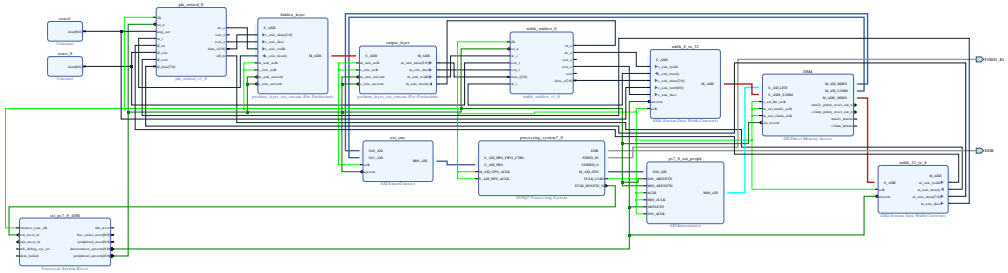


Quire / Exact

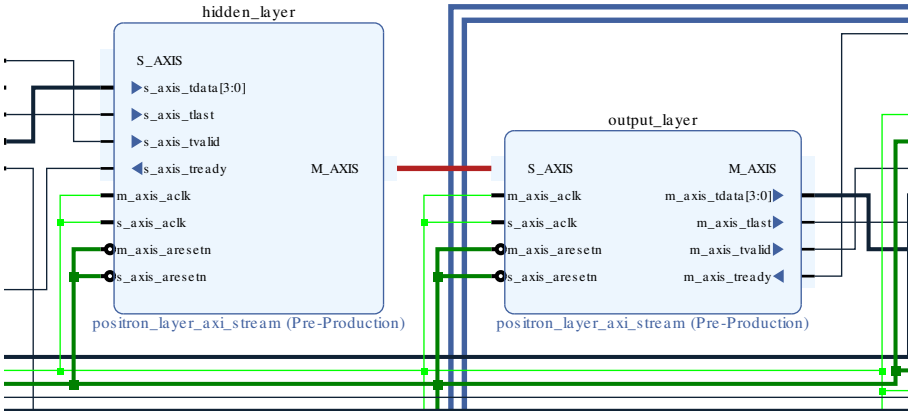


FMA unit, mimicking IEEE754.

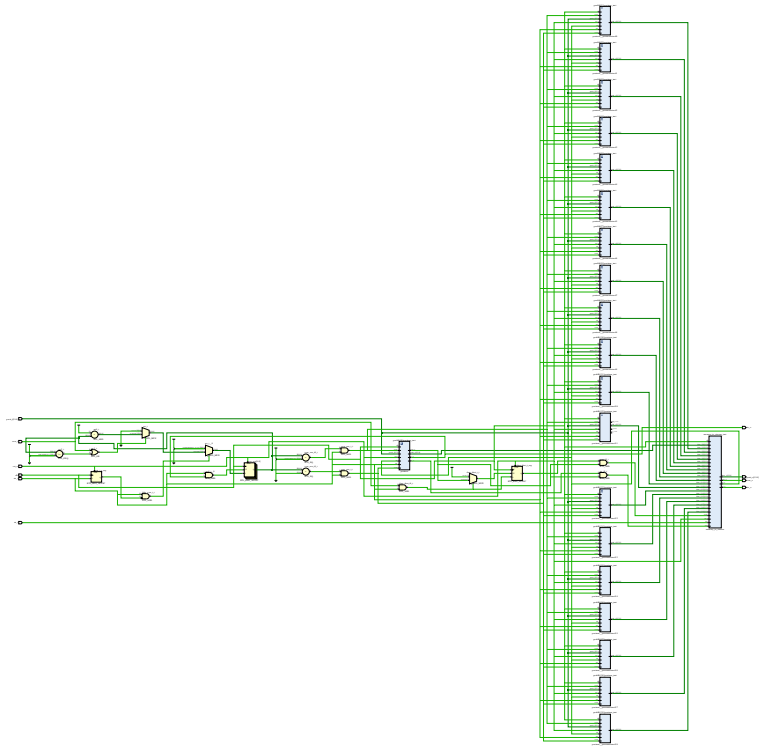
Bottom-Up Development: Fully Connected (FC) Layer



System layout on XC7020 Arty board.

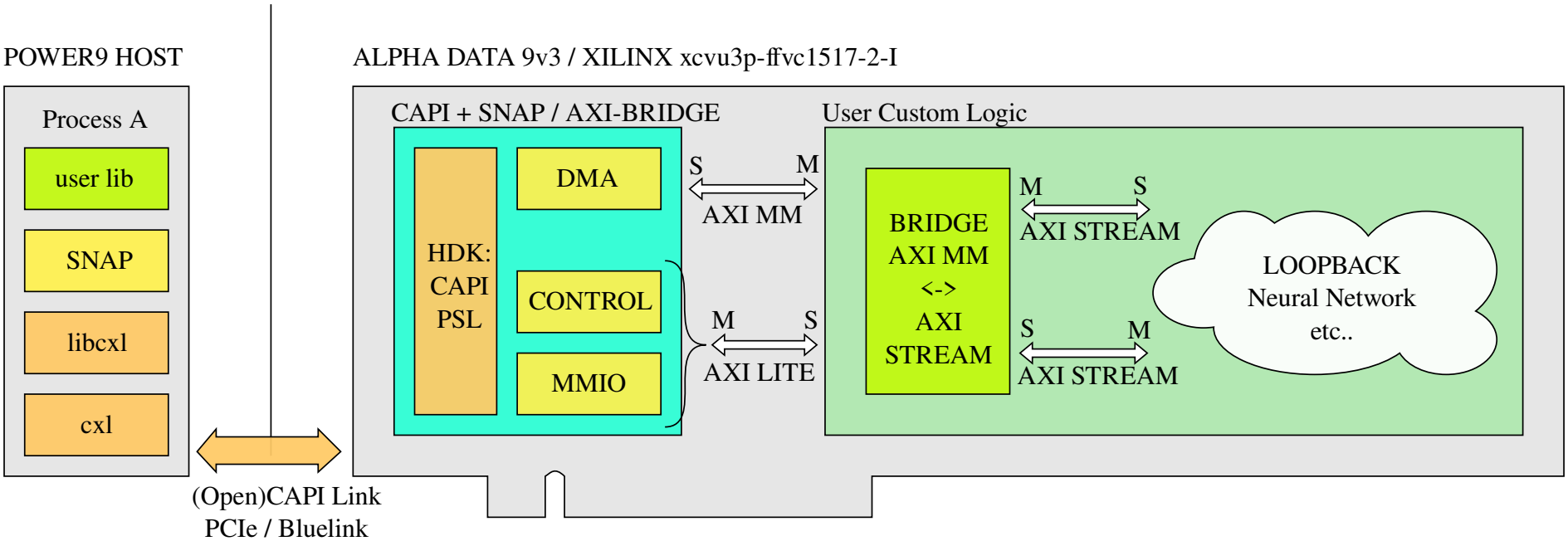


Detailed neural network layers.



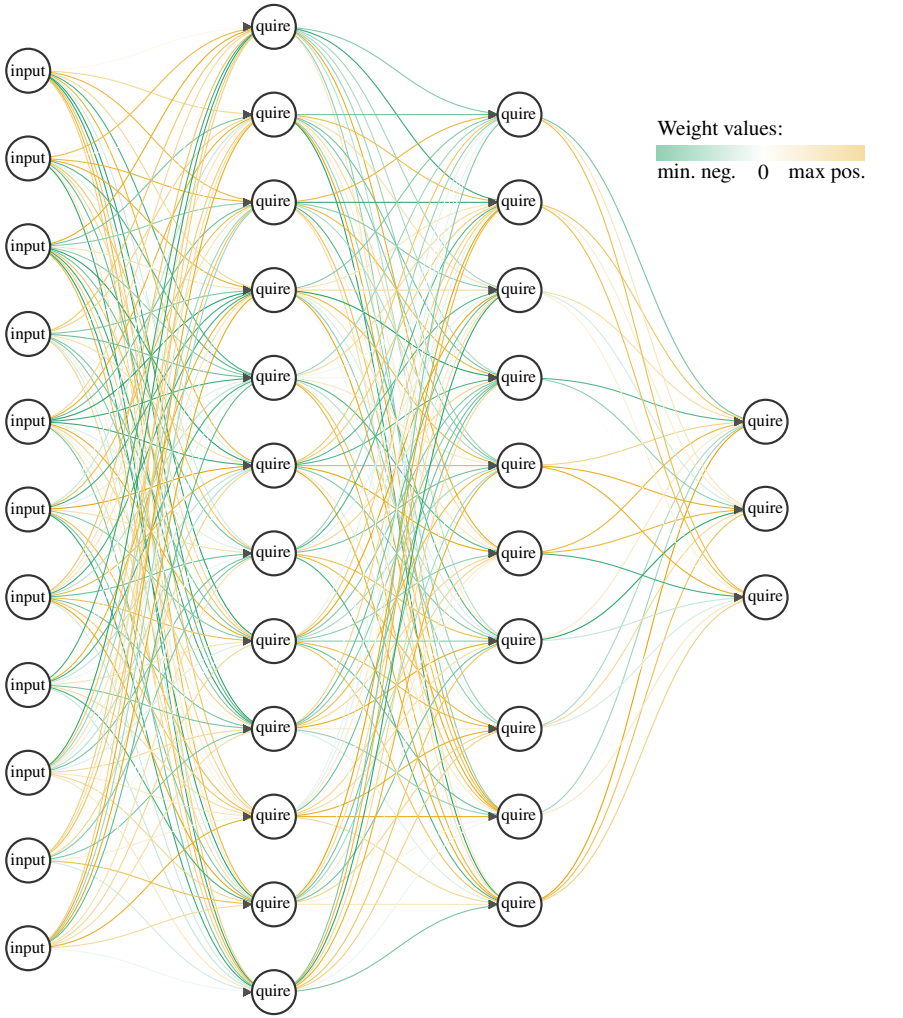
Data aggregation of 20 neurons.

Horizontal Development: PCI-e Performance & Integrity



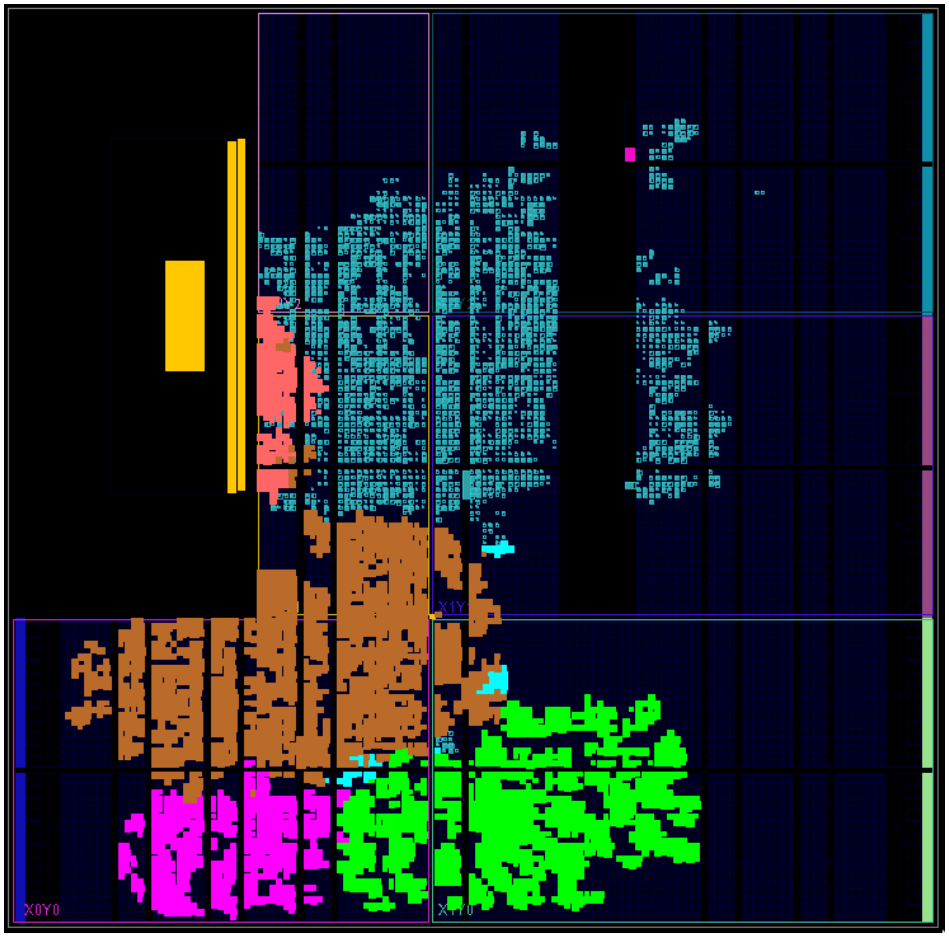
Integration schematic on Power9: testing **data integrity** and **performance** in a pioneering environment. From **loopback** to **neural network**.

MLP Example



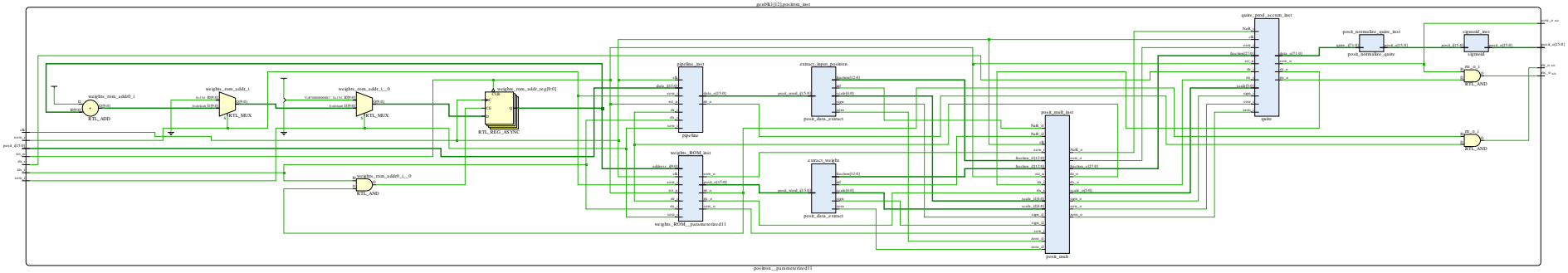
Input Layer $\in \mathbb{R}^{11}$ Hidden Layer $\in \mathbb{R}^{12}$ Hidden Layer $\in \mathbb{R}^{10}$ Output Layer $\in \mathbb{R}^{3}$

Posit MLP in a small FPGA



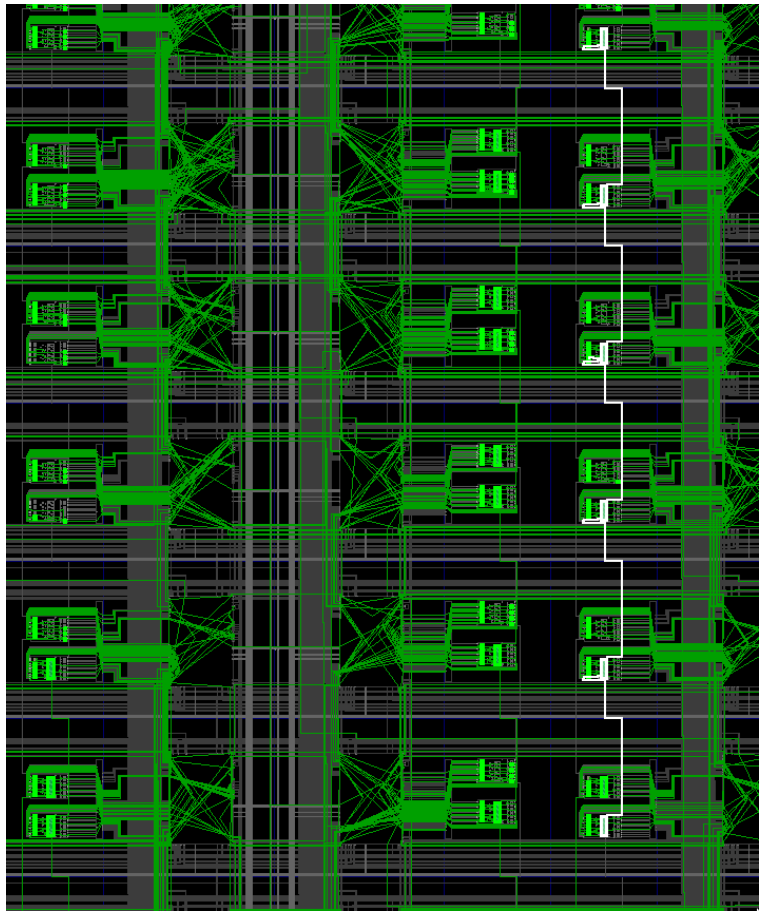
The 2 layers in brown and green of a posit MLP in Arty7020.

Elaborated Positron



Vivado schematic of an elaborated neuron with posits.

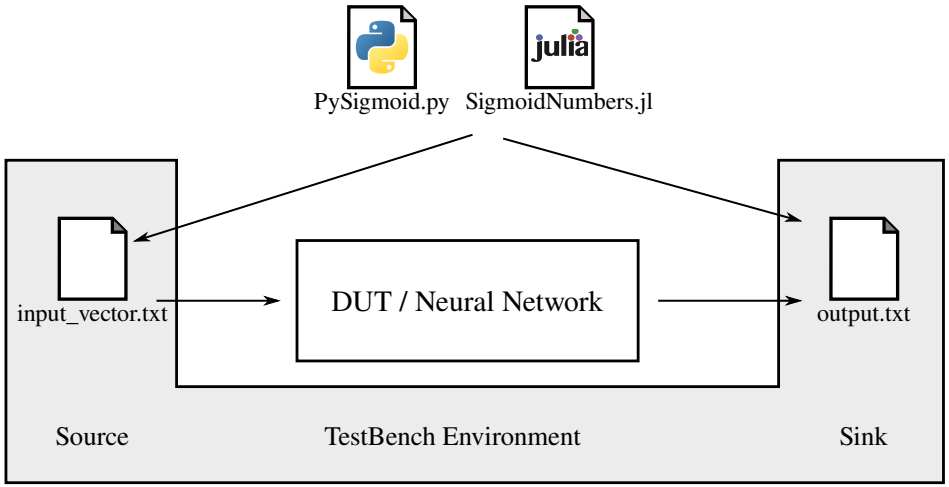
Quire critical path using fast carry chain



Routed view of an MLP hidden layer, with an accumulator carry chain highlighted.

Bottom-Up Development: Verification Methodology

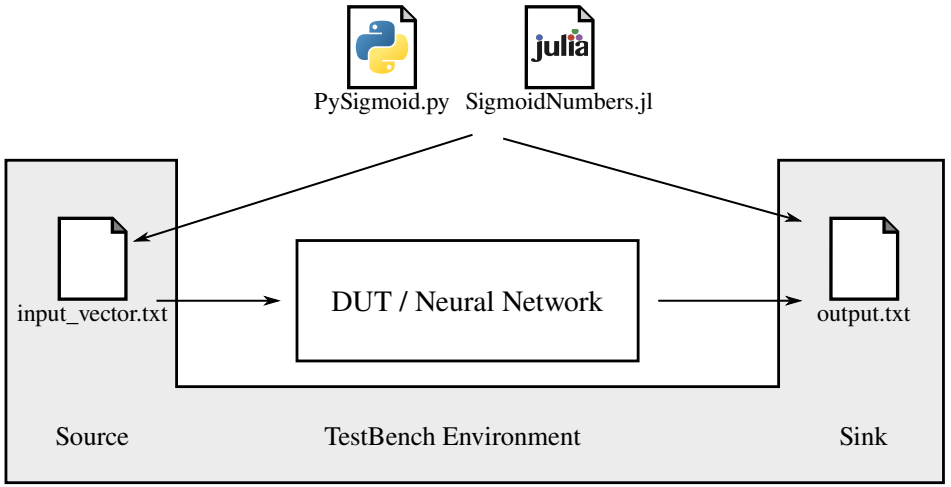
- Bit-to-bit verification vectors are generated and verified using high-level languages supporting posit semantics.
- Source and sink methodology for robust testing.
- PSLSE allows full system emulation of PCIe bursts using identical C code on x86 and Power machines.



Source and sink methodology for module verification, showcasing input generation and output validation.

Bottom-Up Development: Verification Methodology

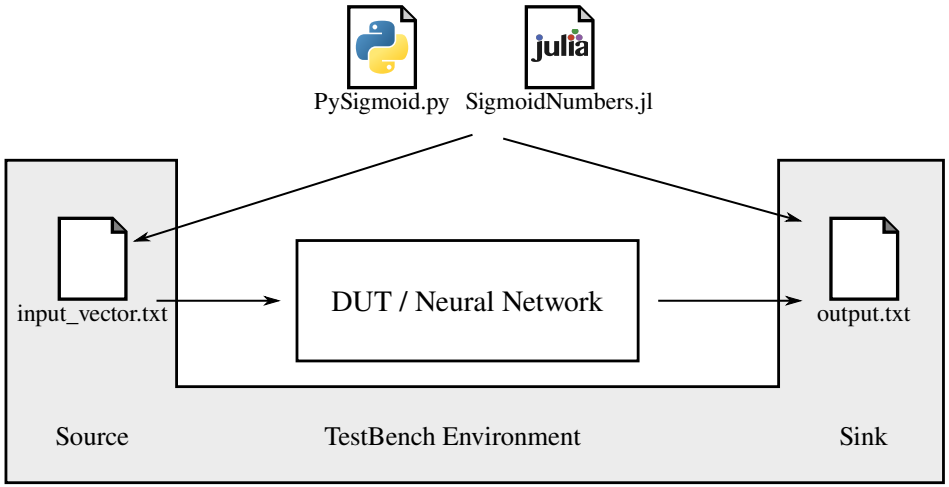
- Bit-to-bit verification vectors are generated and verified using high-level languages supporting posit semantics.
- Source and sink methodology for robust testing.
- PSLSE allows full system emulation of PCIe bursts using identical C code on x86 and Power machines.



Source and sink methodology for module verification, showcasing input generation and output validation.

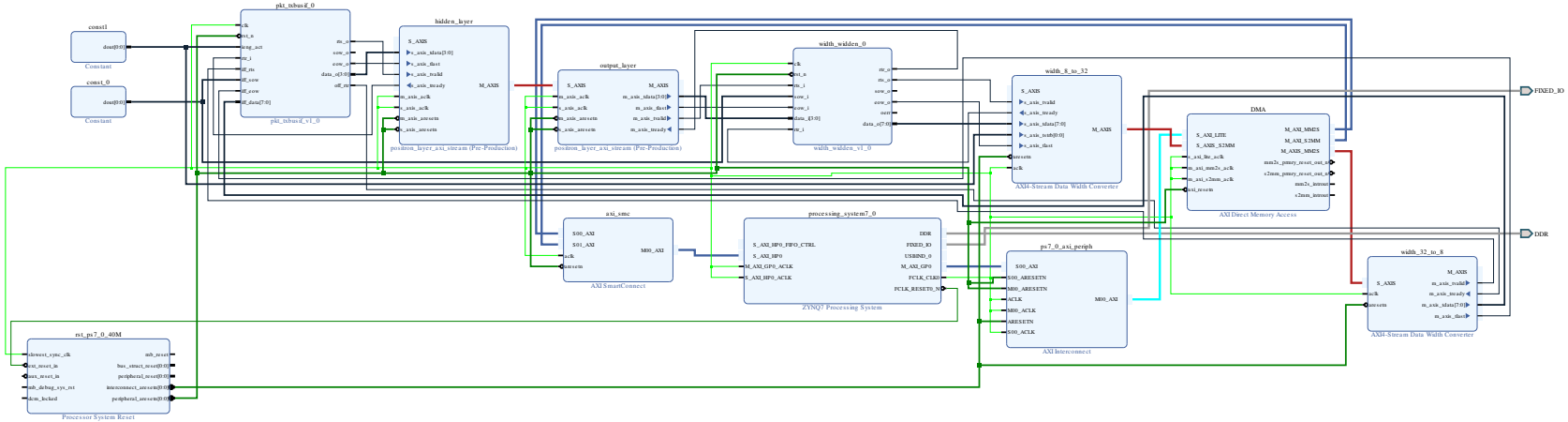
Bottom-Up Development: Verification Methodology

- Bit-to-bit verification vectors are generated and verified using high-level languages supporting posit semantics.
- Source and sink methodology for robust testing.
- PSLSE allows full system emulation of PCIe bursts using identical C code on x86 and Power machines.

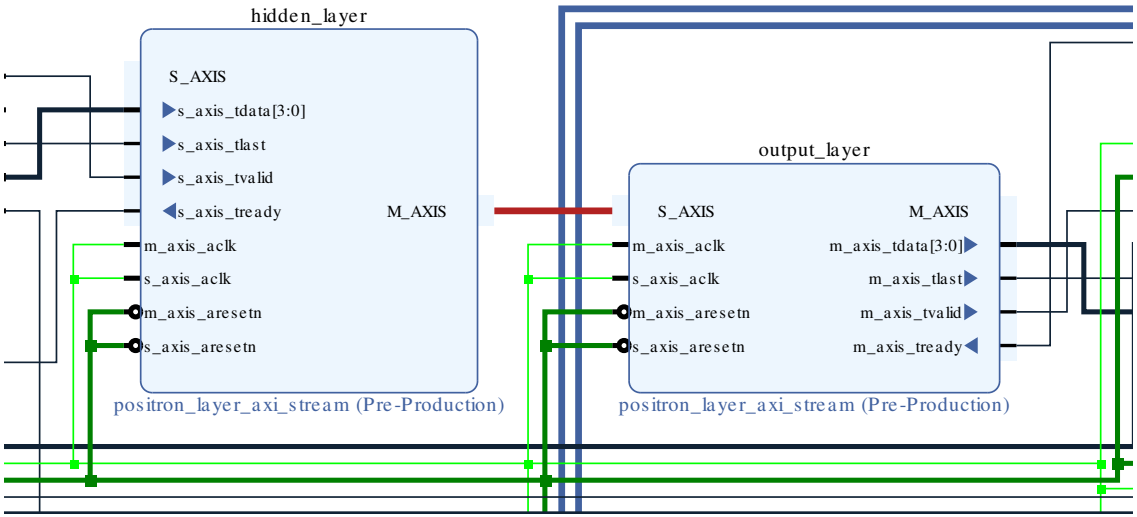


Source and sink methodology for module verification, showcasing input generation and output validation.

FC Layer Development: Vivado GUI Integration



System layout on XC7020 Arty board.



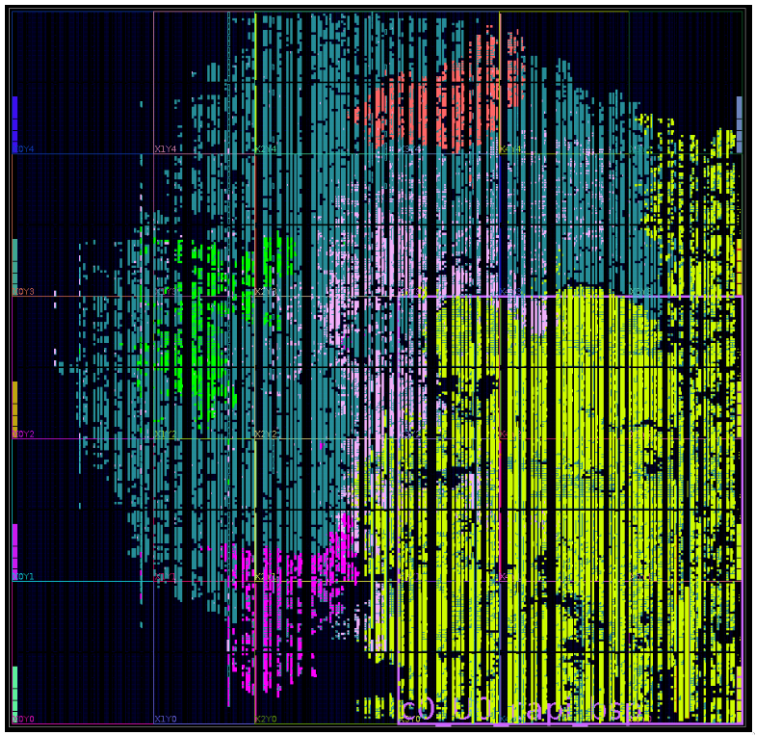
Detailed neural network layers.

FPS Calculation and Example Configuration

- Frames per Second (FPS) is a throughput metric in the context of image processing
- let n be bitwidth, c occupancy ratio

$$\text{FPS}(n, c) \approx \frac{12 \times c \times 10^9}{784 \times \frac{n}{8}}$$

- Example features 16 MLPS of 8-bit posit
- one quarter of the link is utilized
- Example FPS = $FPS(8, 0.25) \approx 3.8 \times 10^6$



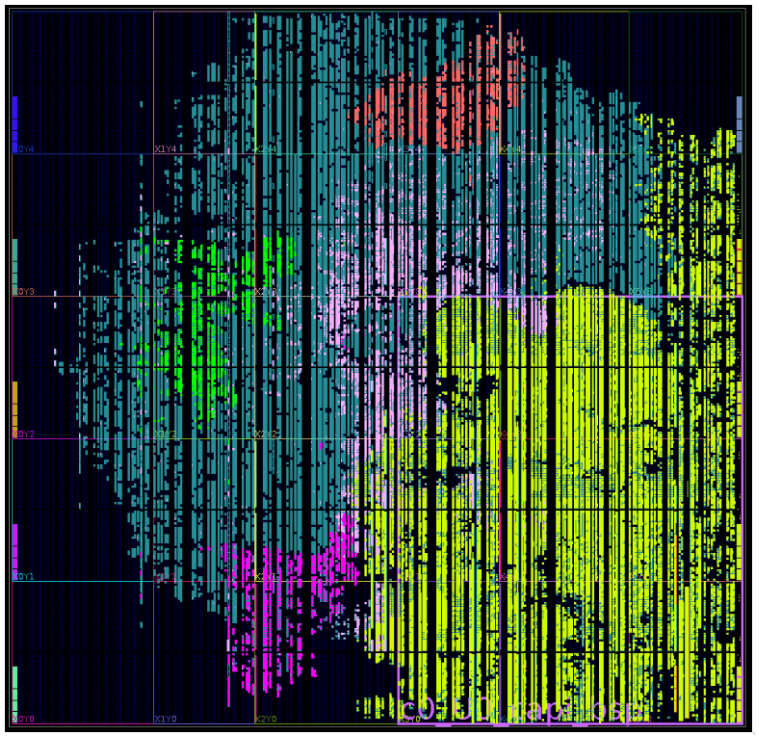
16 MLPs on VU3P FPGA, utilizing 25% of bandwidth.

FPS Calculation and Example Configuration

- Frames per Second (FPS) is a throughput metric in the context of image processing
- let n be bitwidth, c occupancy ratio

$$\text{FPS}(n, c) \approx \frac{12 \times c \times 10^9}{784 \times \frac{n}{8}}$$

- Example features 16 MLPS of 8-bit posit
- one quarter of the link is utilized
- Example FPS = $FPS(8, 0.25) \approx 3.8 \times 10^6$



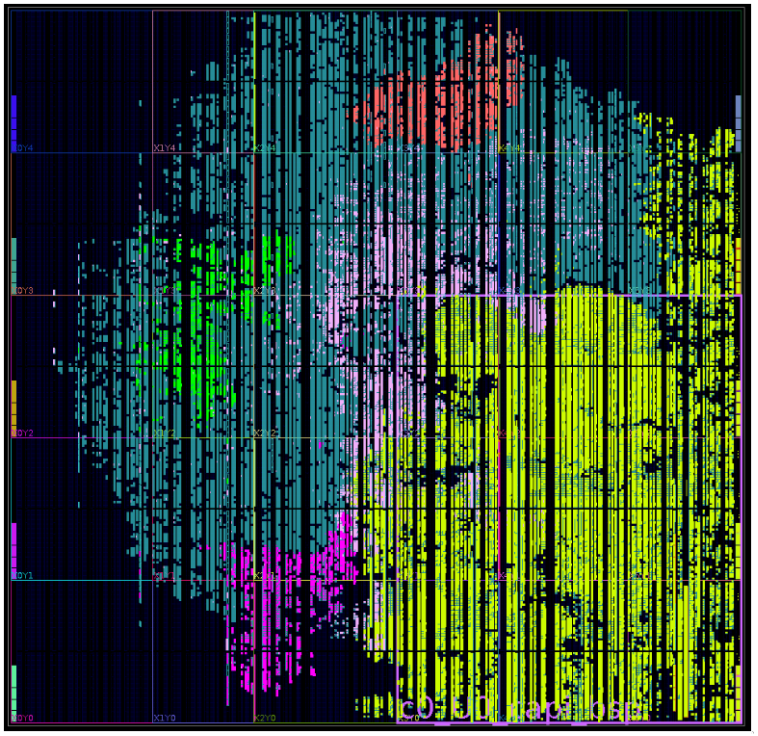
16 MLPs on VU3P FPGA, utilizing 25% of bandwidth.

FPS Calculation and Example Configuration

- Frames per Second (FPS) is a throughput metric in the context of image processing
- let n be bitwidth, c occupancy ratio

$$\text{FPS}(n, c) \approx \frac{12 \times c \times 10^9}{784 \times \frac{n}{8}}$$

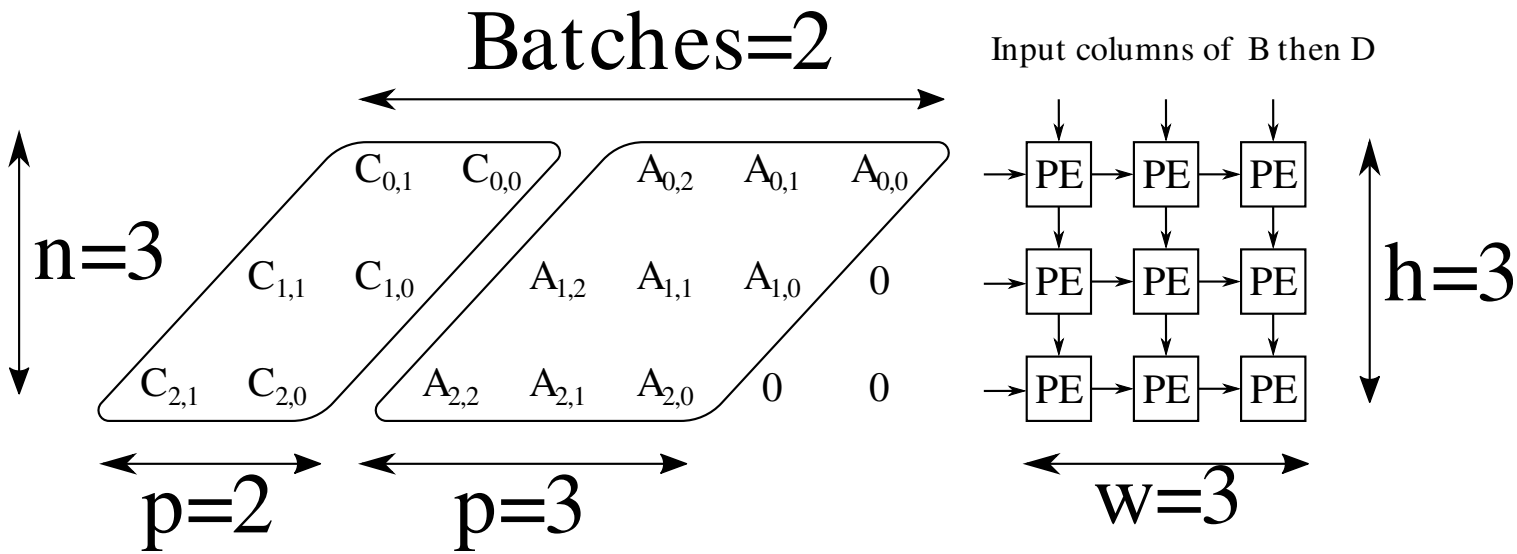
- Example features 16 MLPS of 8-bit posit
- one quarter of the link is utilized
- Example FPS = $FPS(8, 0.25) \approx 3.8 \times 10^6$



16 MLPs on VU3P FPGA, utilizing 25% of bandwidth.

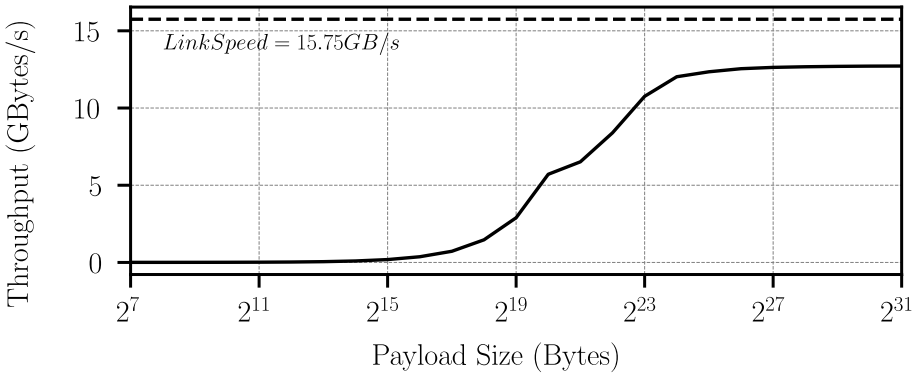
Systolic Array Generator: Half Speed Sink Down (HSSD)

- **Scaling:** Local connectivity minimizes global routing.
- **Performance:** Stalling-free output and batch processing of independent GEMMs.

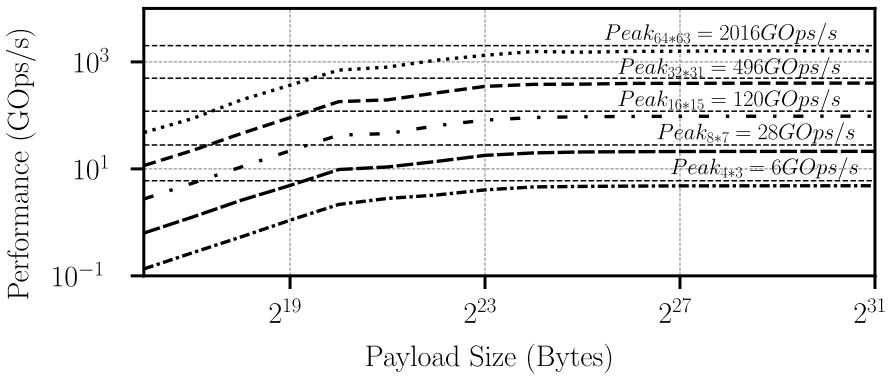


batch processing using HSSD.

Evaluation: Throughput and Performance



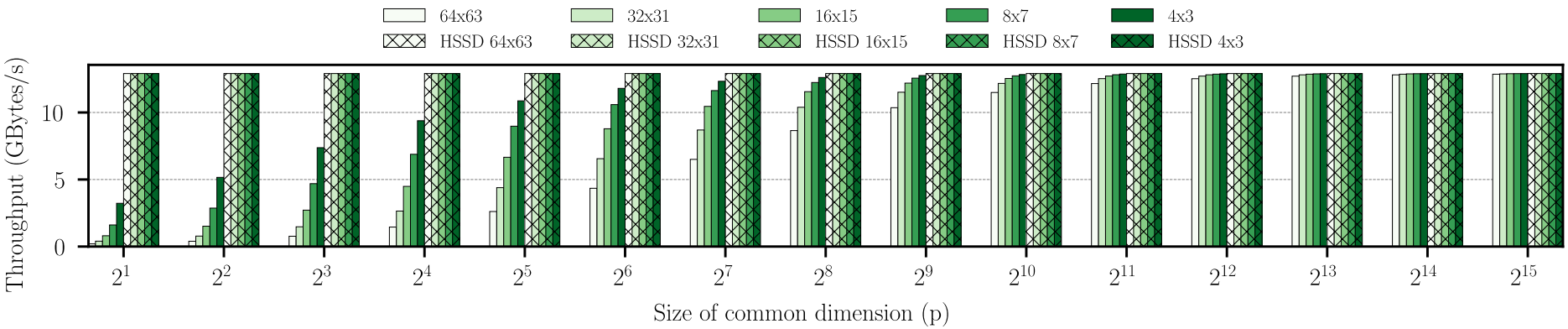
Measured vs. theoretical Throughput



Measured vs. theoretical Performances

Evaluation: HSSD impact

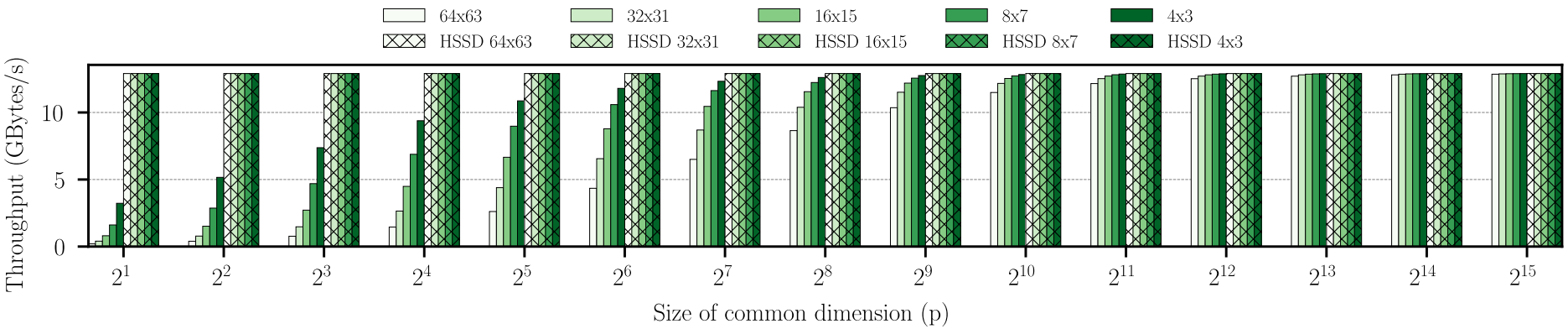
- Payload of 1GB.
- Sweep p the "Common dimension" in $A_{m\times p} \times B_{p\times n} = C_{m\times n}$.
- **HSSD saturates** the link for all sizes



HSSD impact for 2^{30} Bytes payload across dimension sizes (p).

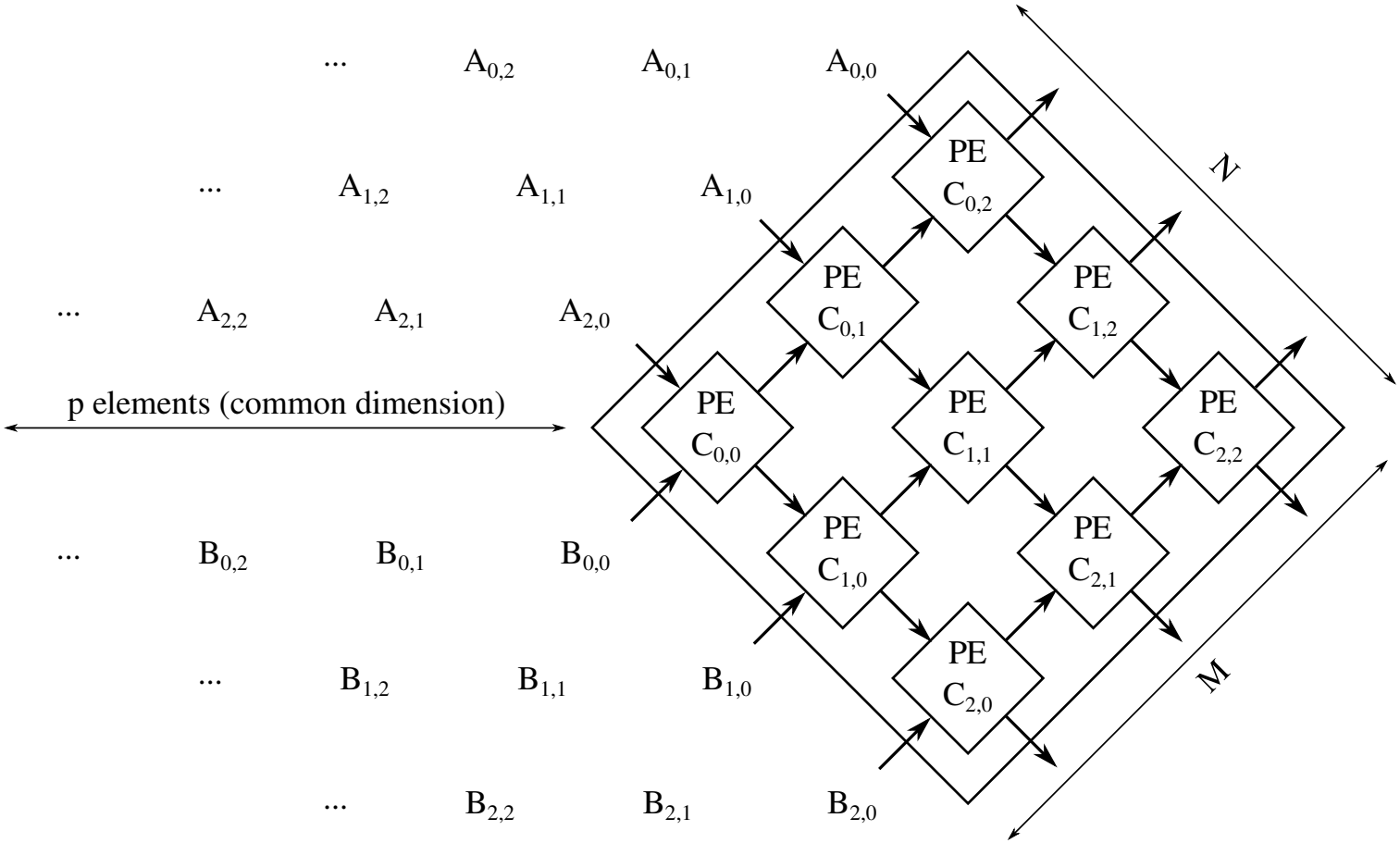
Evaluation: HSSD impact

- Payload of 1GB.
- Sweep p the "Common dimension" in $A_{m\times p} \times B_{p\times n} = C_{m\times n}$.
- **HSSD saturates** the link for all sizes

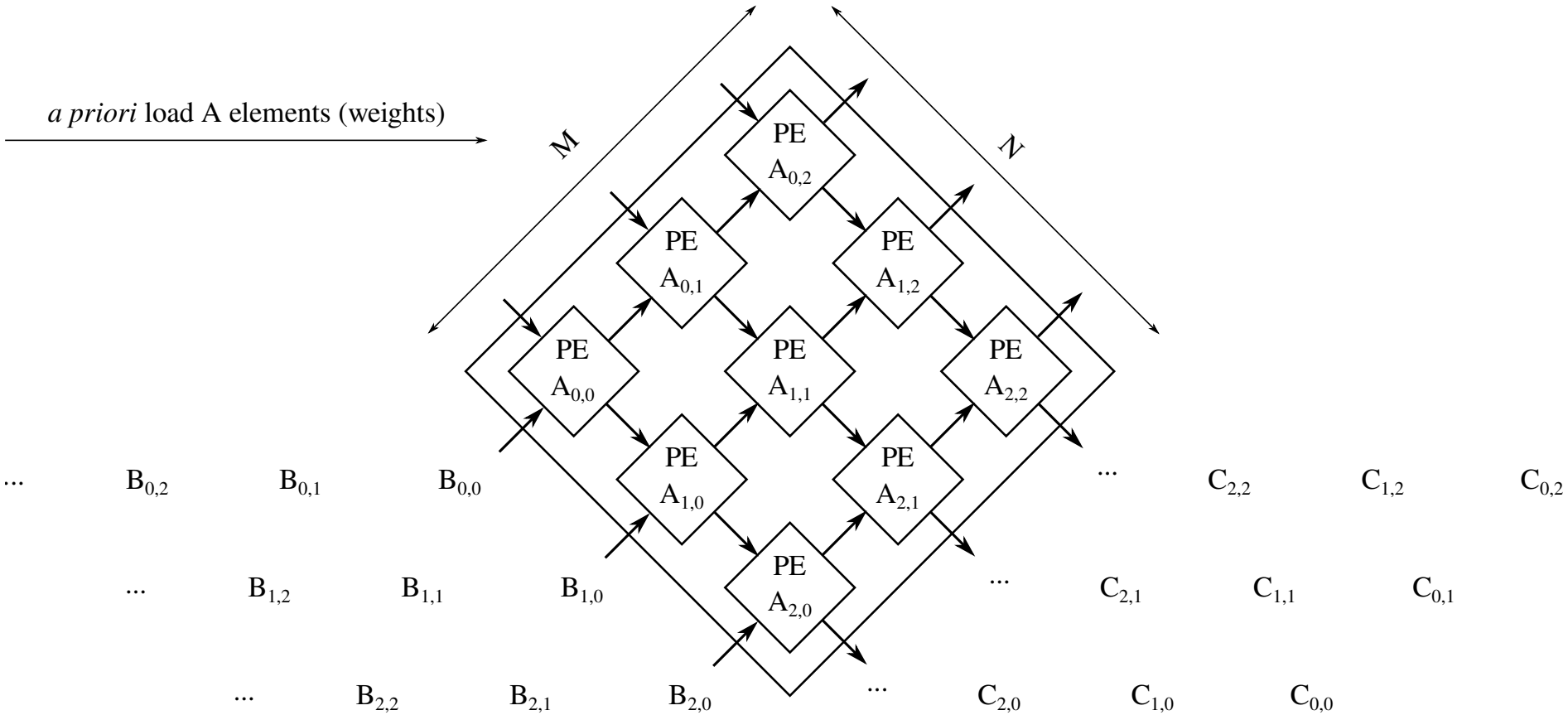


HSSD impact for 2^{30} Bytes payload across dimension sizes (p).

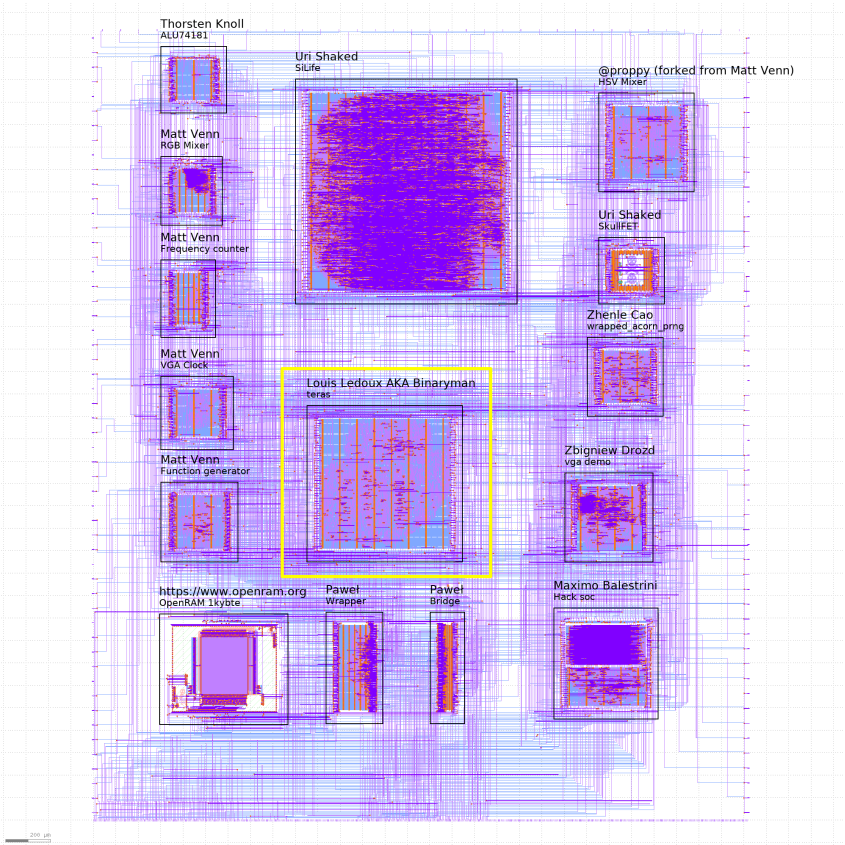
Output-Stationary Systolic Array



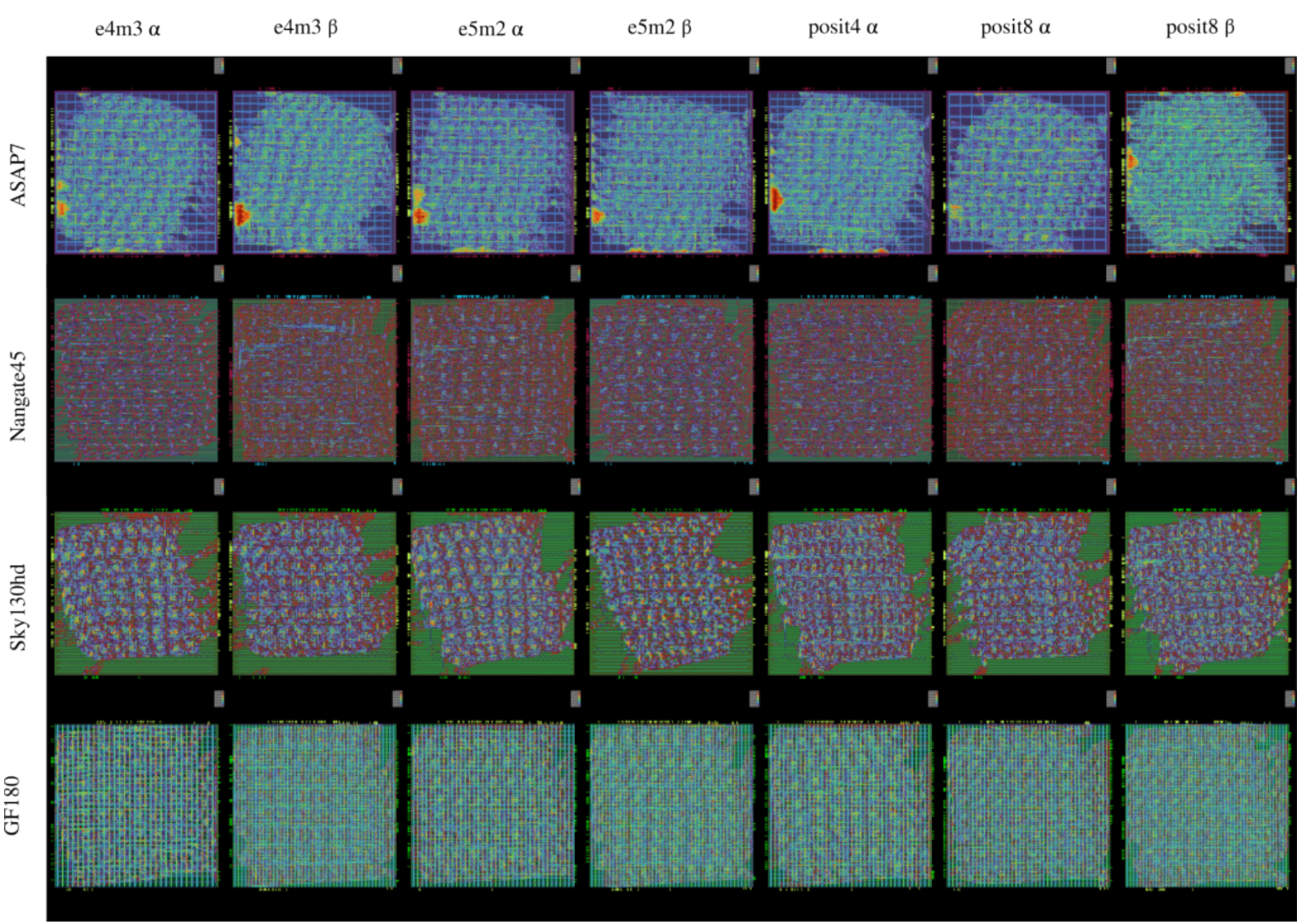
Weight-Stationary Systolic Array



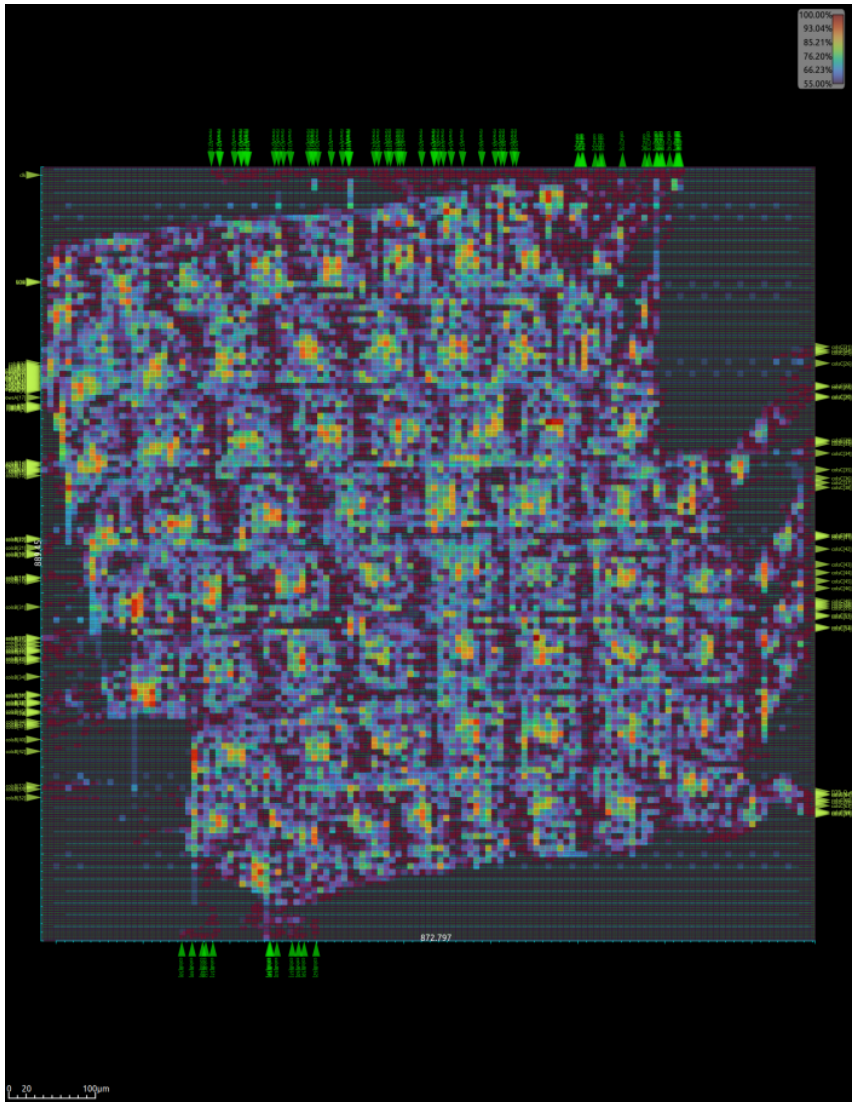
Multi-Project Waffer Tapeout



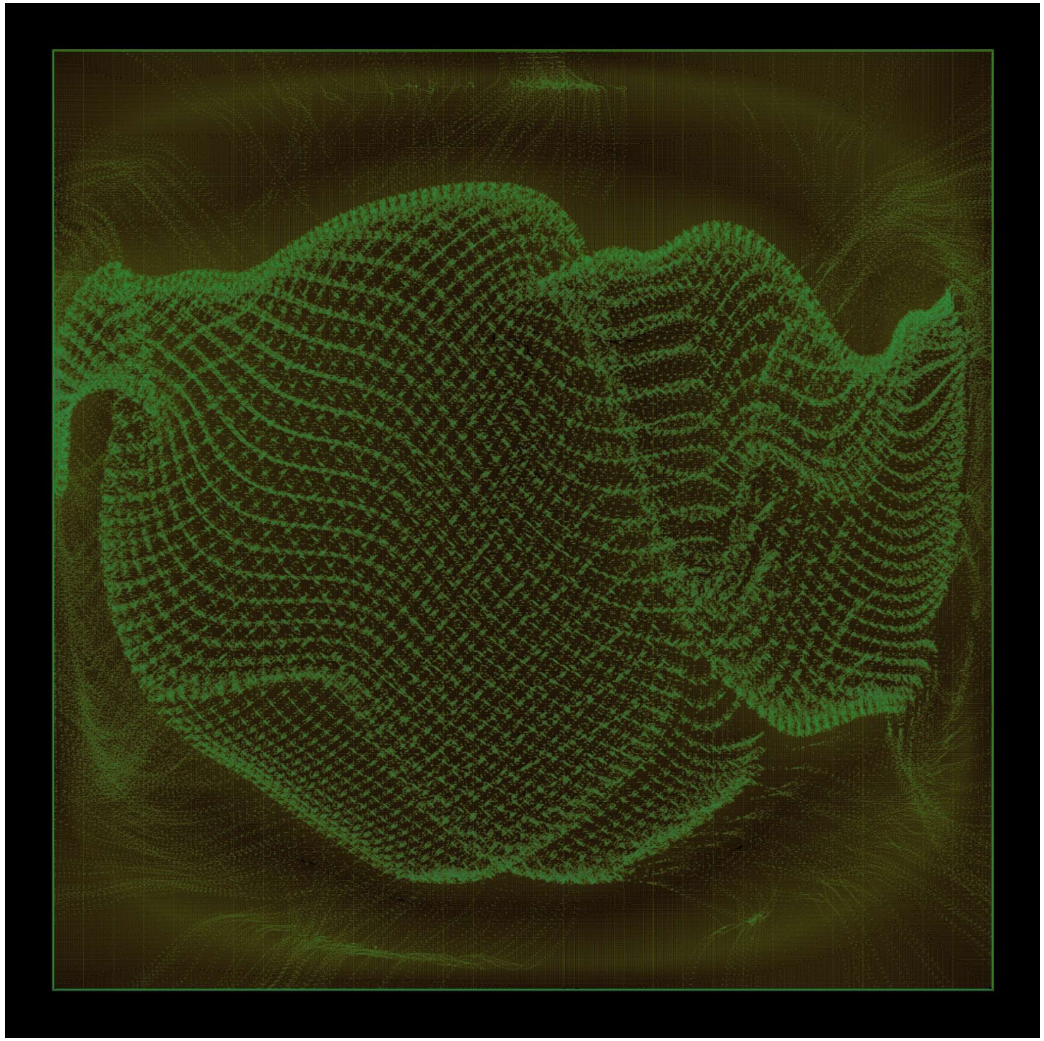
Recent Re-evaluation on smaller FP formats on ASIC



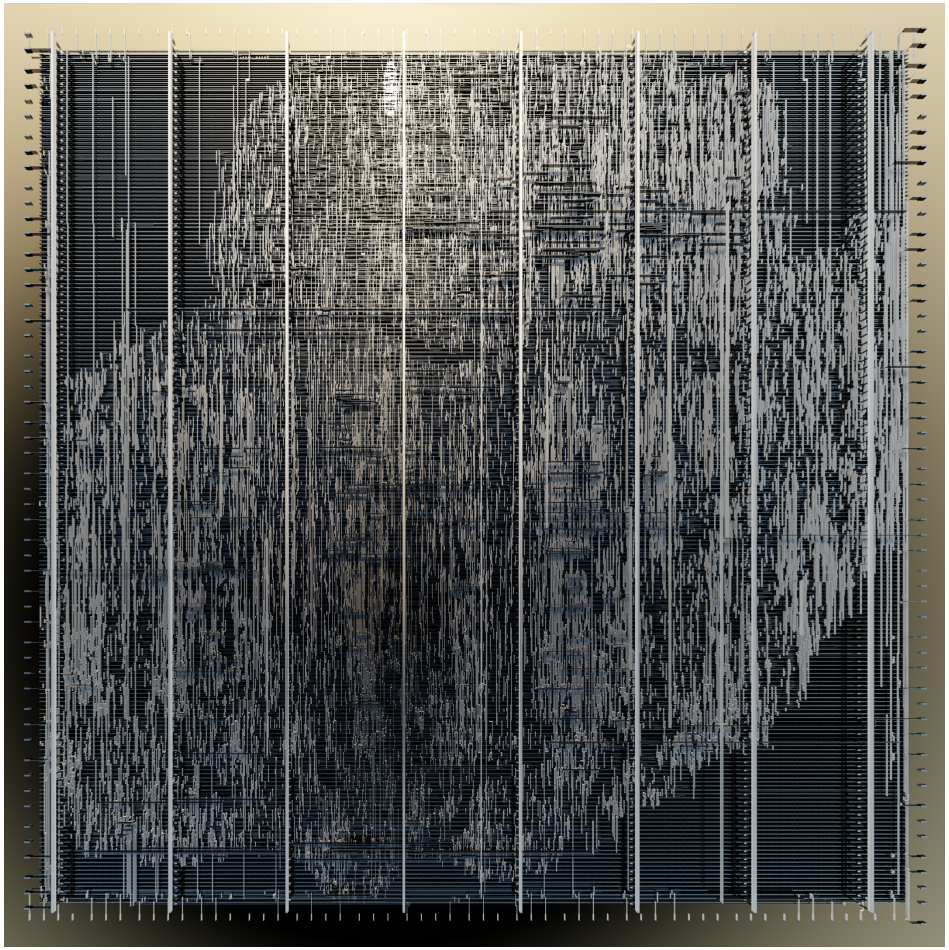
SA 8x8 e4m3 rulers congestion heatmap



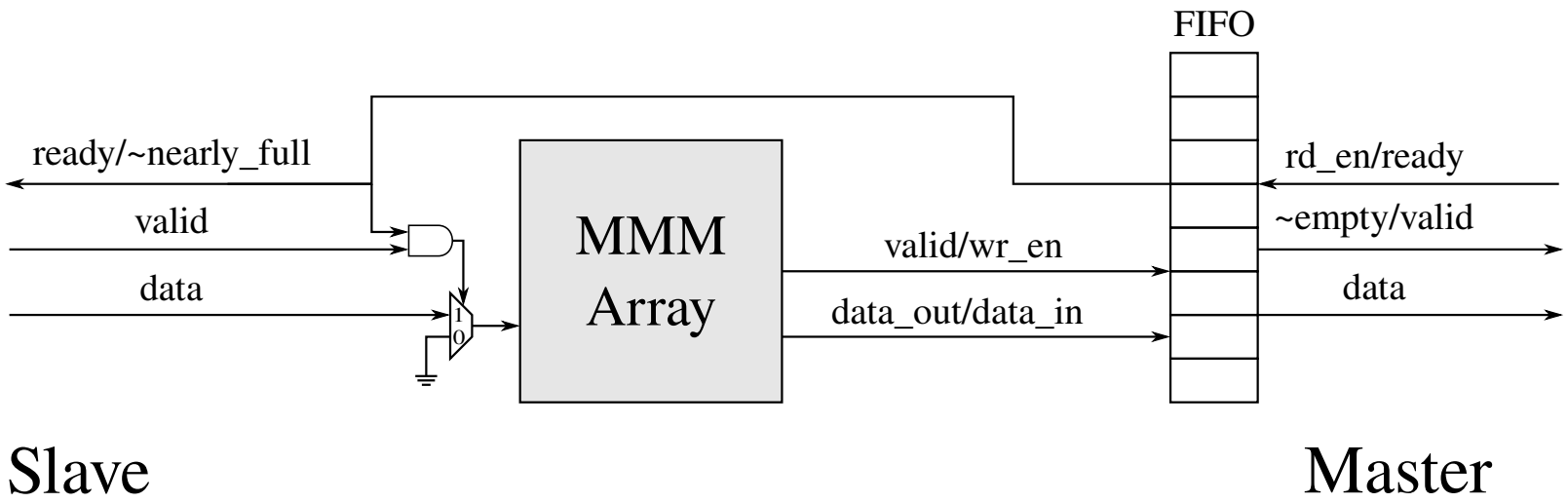
Placement with eletrostatic field of 64x64 ; <1mm2 e4m3



Raytracing top view SA 4x3 posit<8,0>



CAPI backpressure design



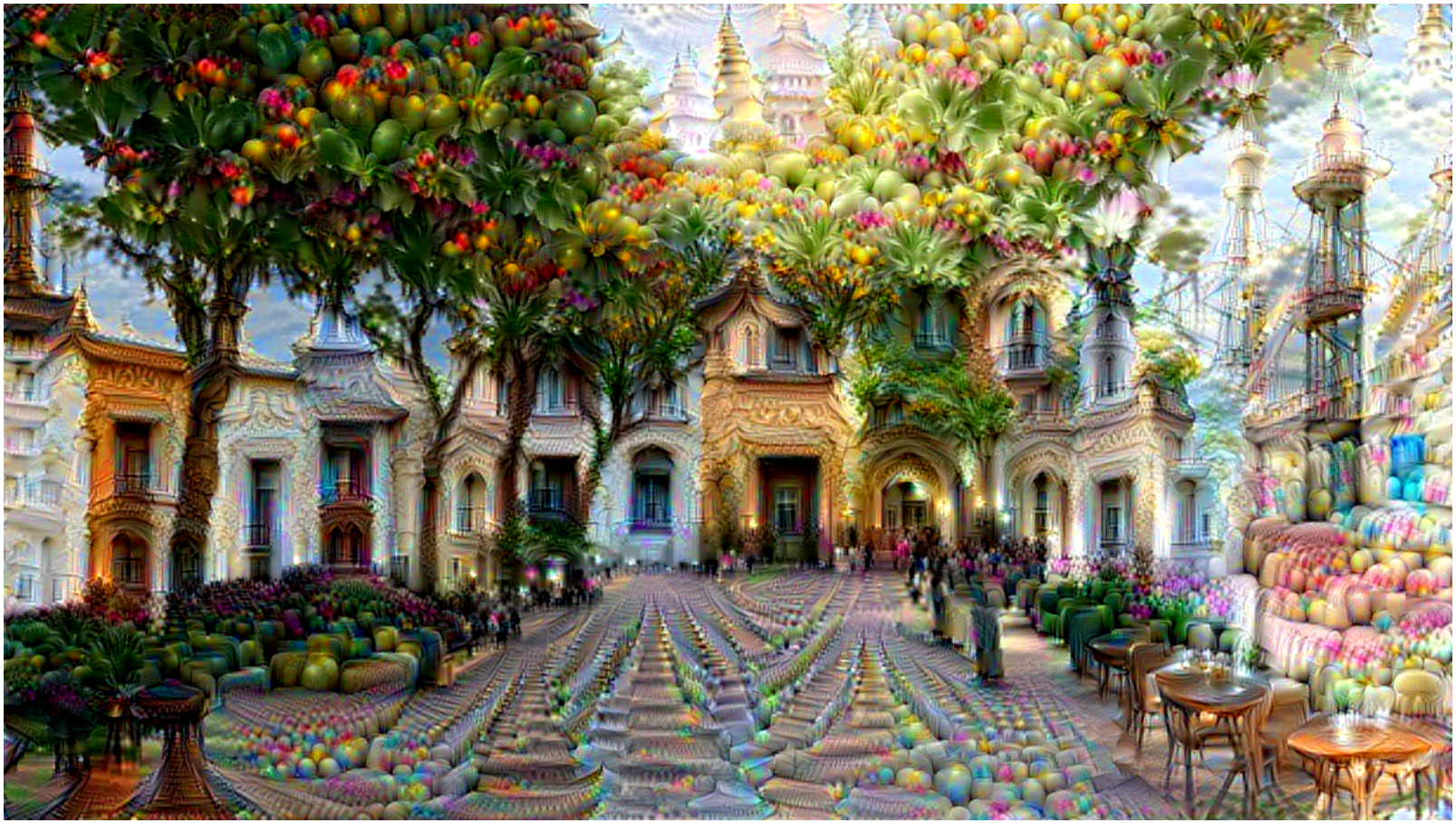
Any code works without modification: Santa Cruz case

A photo of barrio de Santa cruz.

Any code works without modification: Santa Cruz case

Deepdream iteration 1.

Any code works without modification: Santa Cruz case



Deepdream iteration 2.

Any code works without modification: Santa Cruz case

Deepdream iteration 3.

Any code works without modification: Santa Cruz case

Deepdream iteration 4.

Any code works without modification: Santa Cruz case

Deepdream iteration 5.

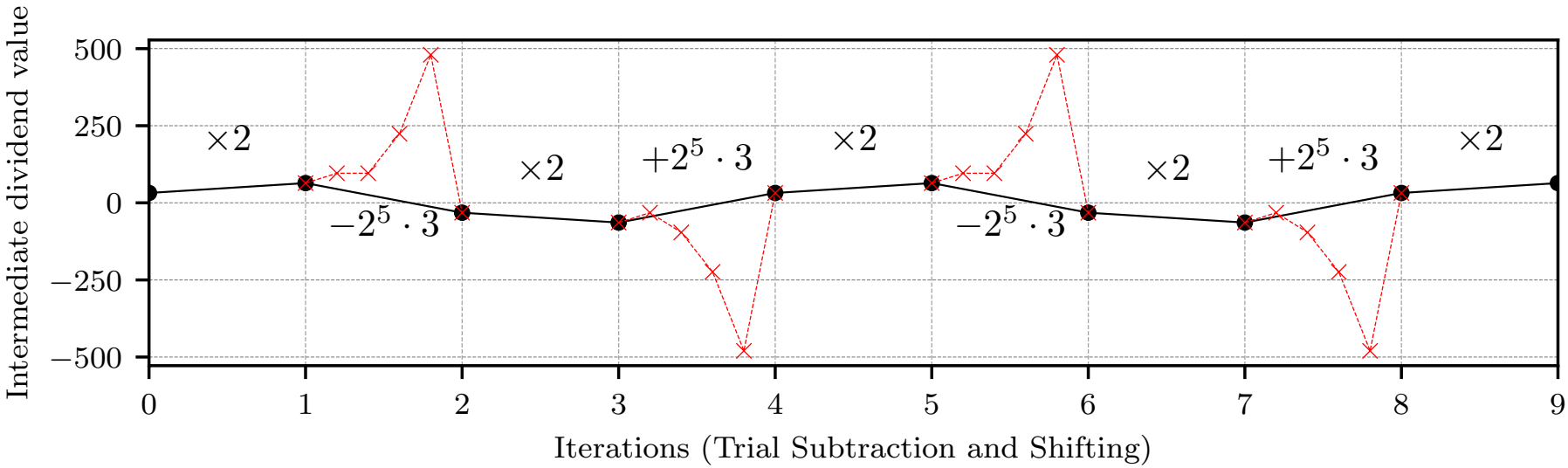
Any code works without modification: Santa Cruz case

Deepdream iteration 6.

Any code works without modification: Santa Cruz case

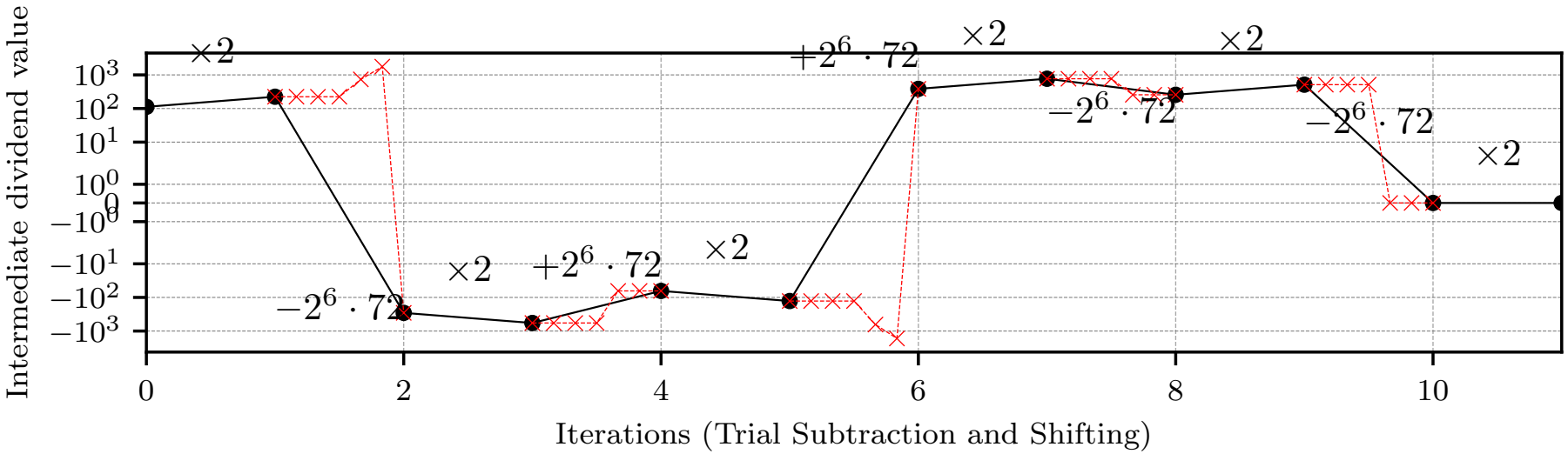
Deepdream iteration 7.

Division Algorithm Contribtion: 32/3 5-bit



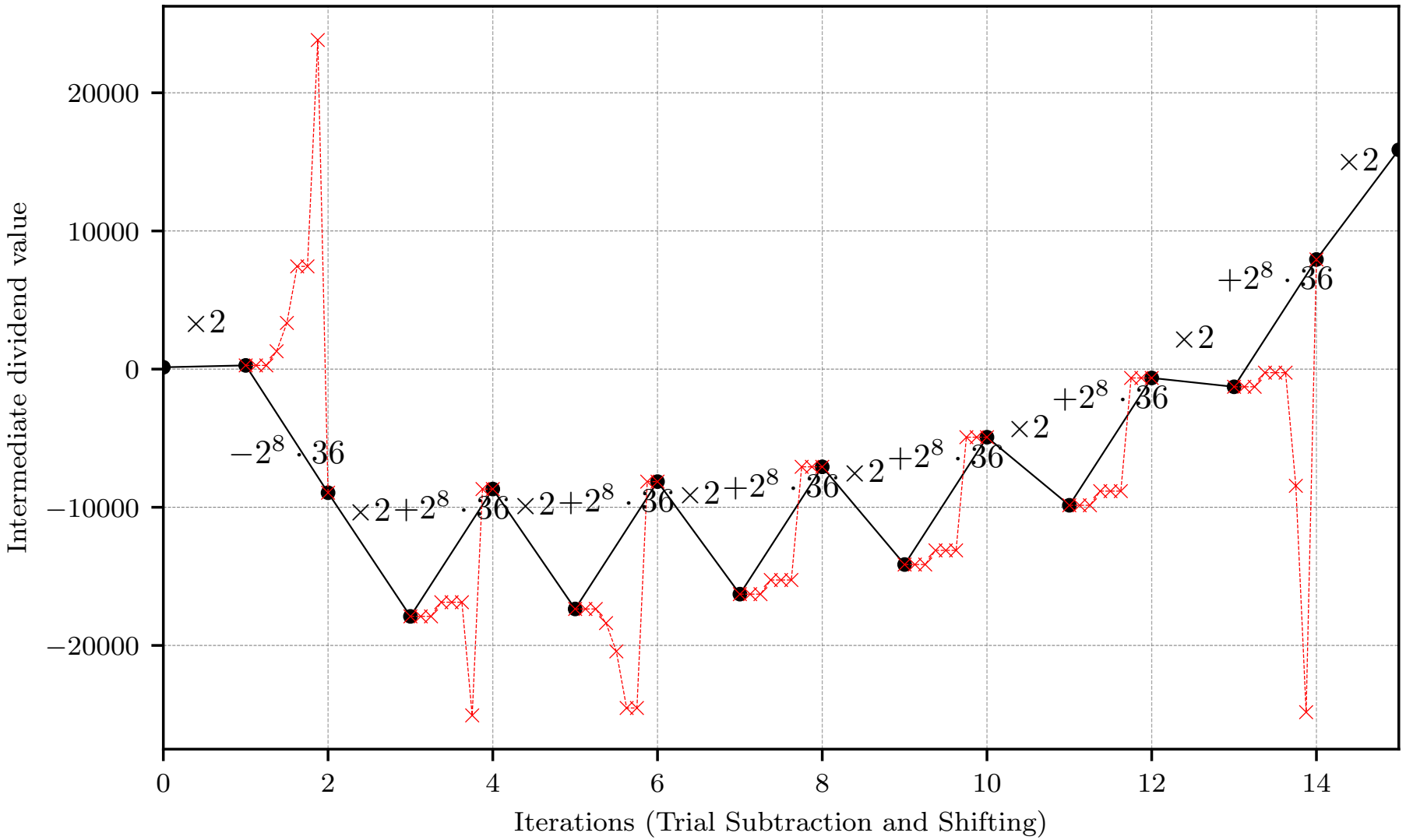
32 divided by 3 on 5-bit.

Division Algorithm Contribtion: 112/-72 6-bit (log scale)

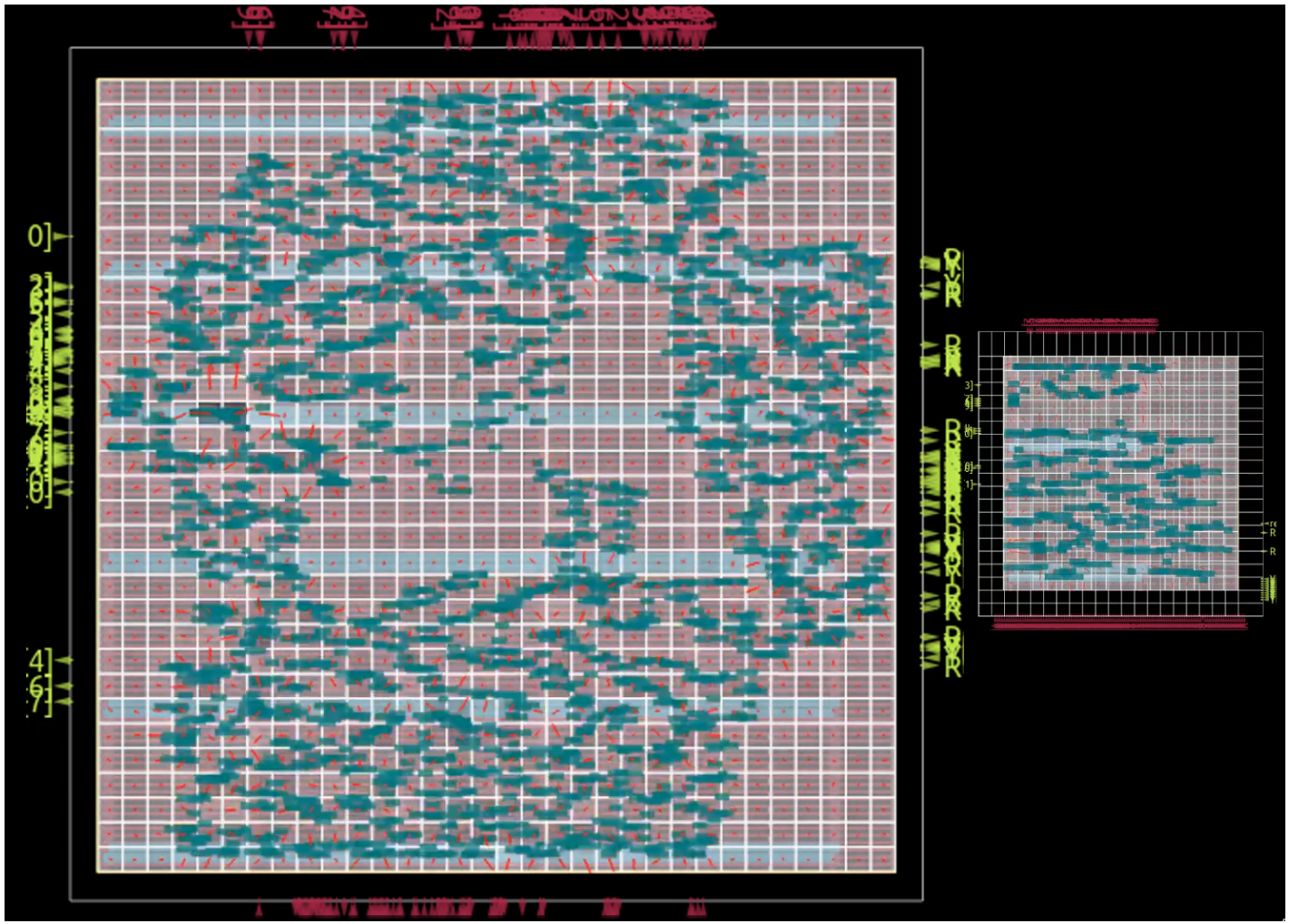


112 divided by -72 on 6-bit.

Division Algorithm Contribtion: 132/36 8-bit

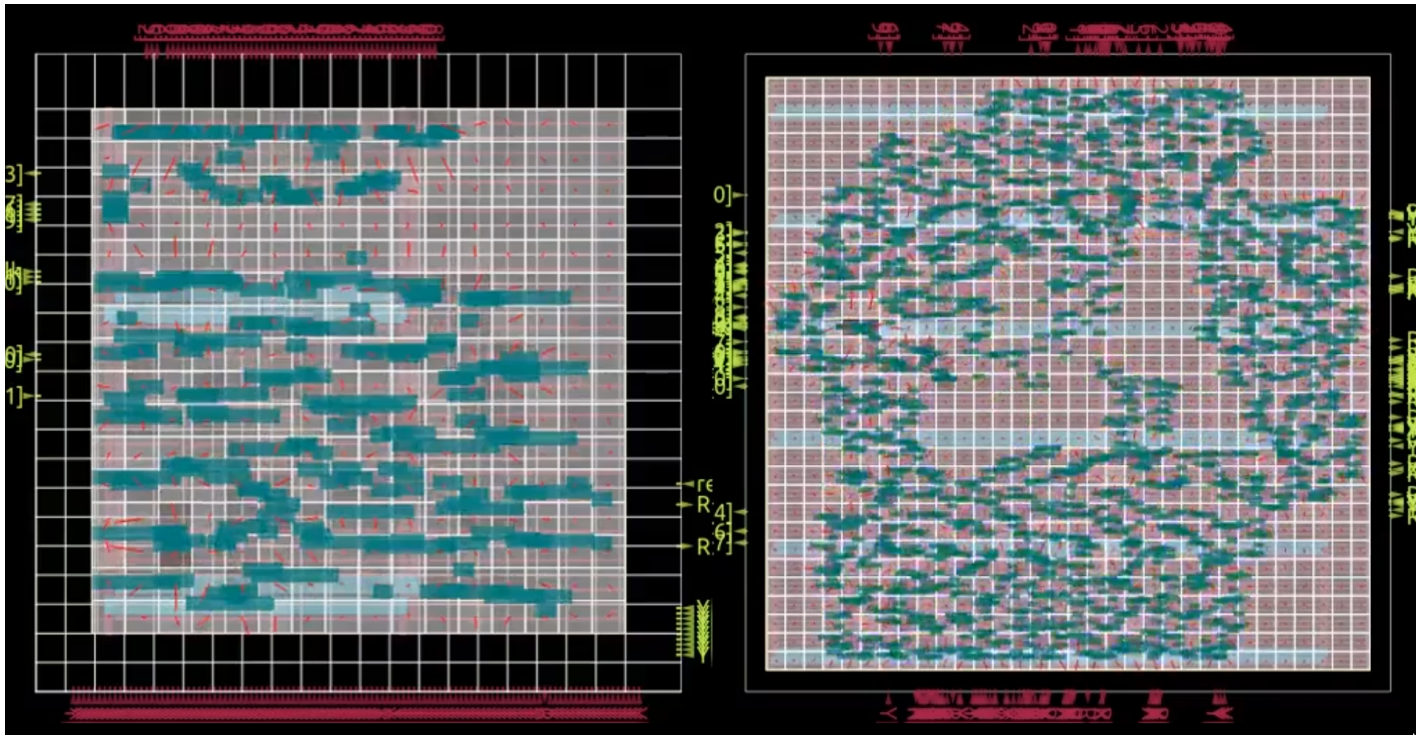


Posit Division layouts: scaled



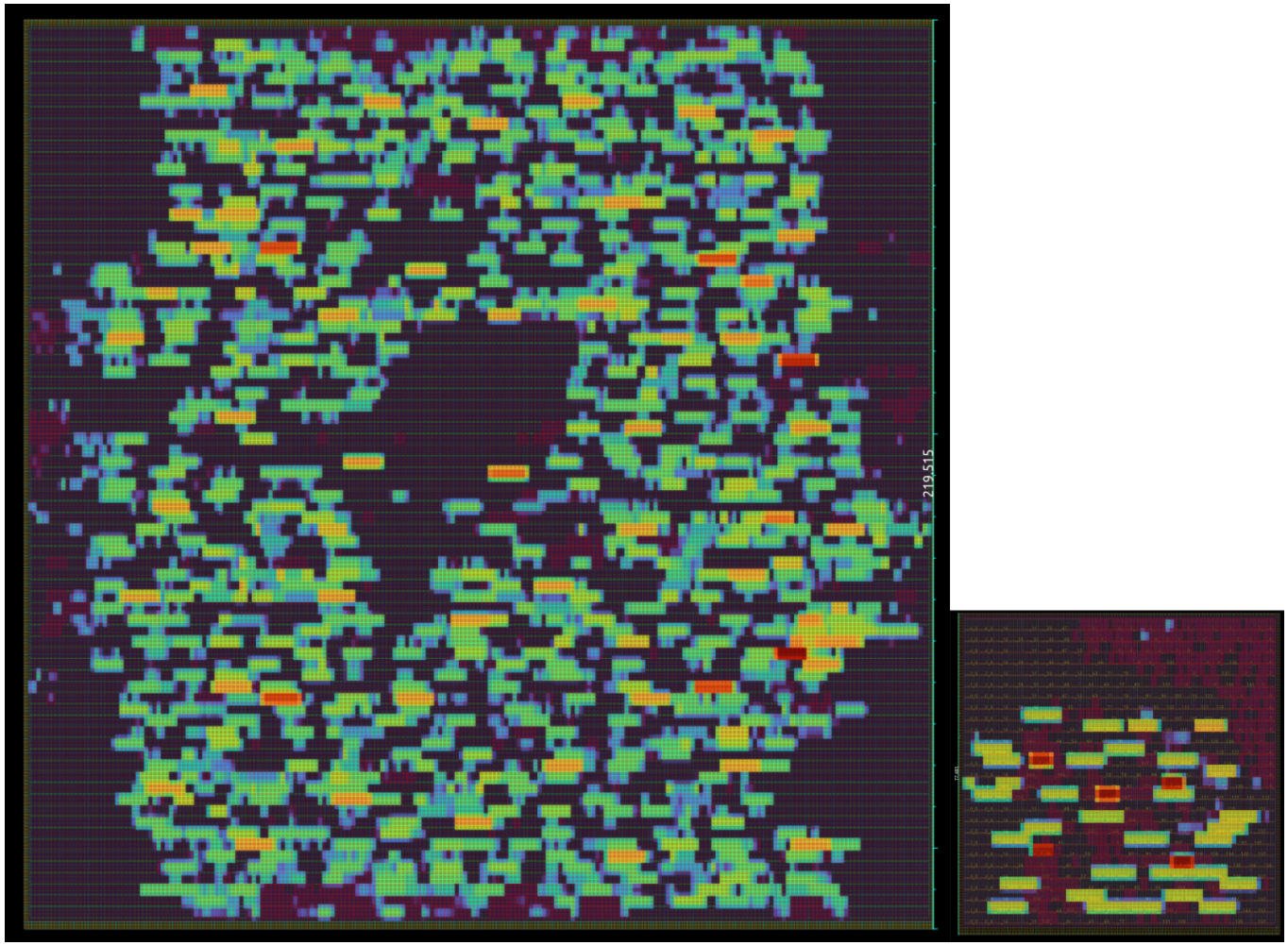
Posit64 Non-Restoring division: baseline (left) and 2-bit serial adder (right).

Posit Division layouts: not scaled



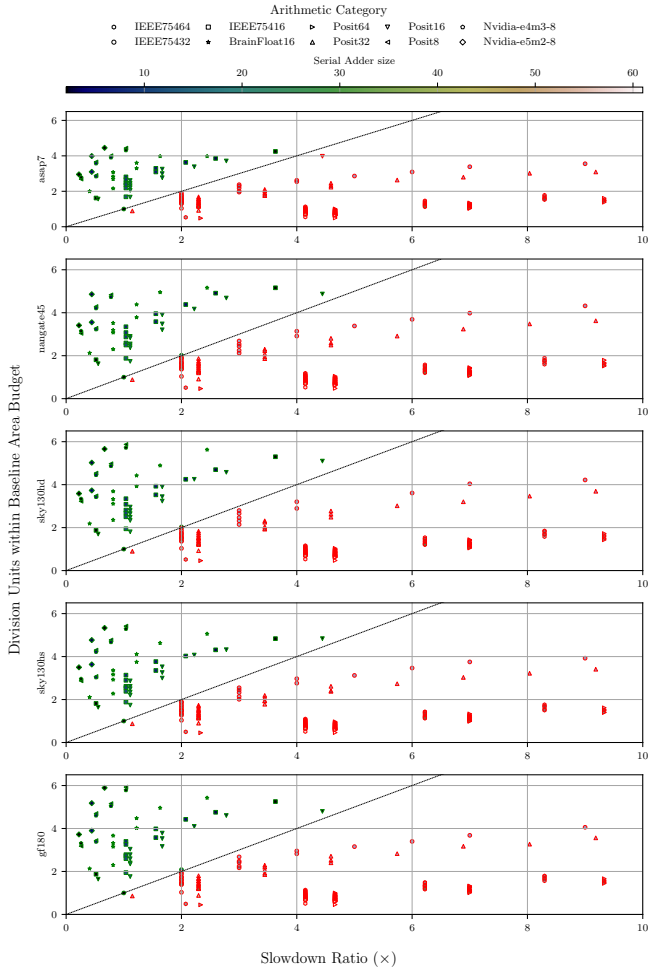
Posit64 Non-Restoring division: baseline (right) and 2-bit serial adder (left).

Posit Division layouts: scaled and heatmap



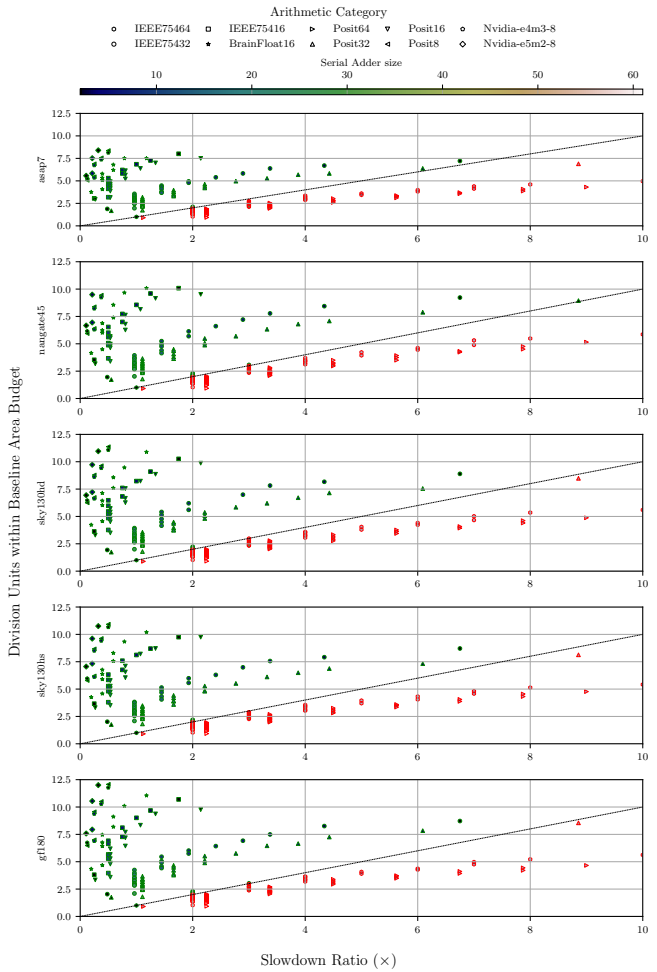
Posit64 Non-Restoring division: baseline (left) and 2-bit serial adder (right).

SAP results against IEEE754: Area, 32-bit



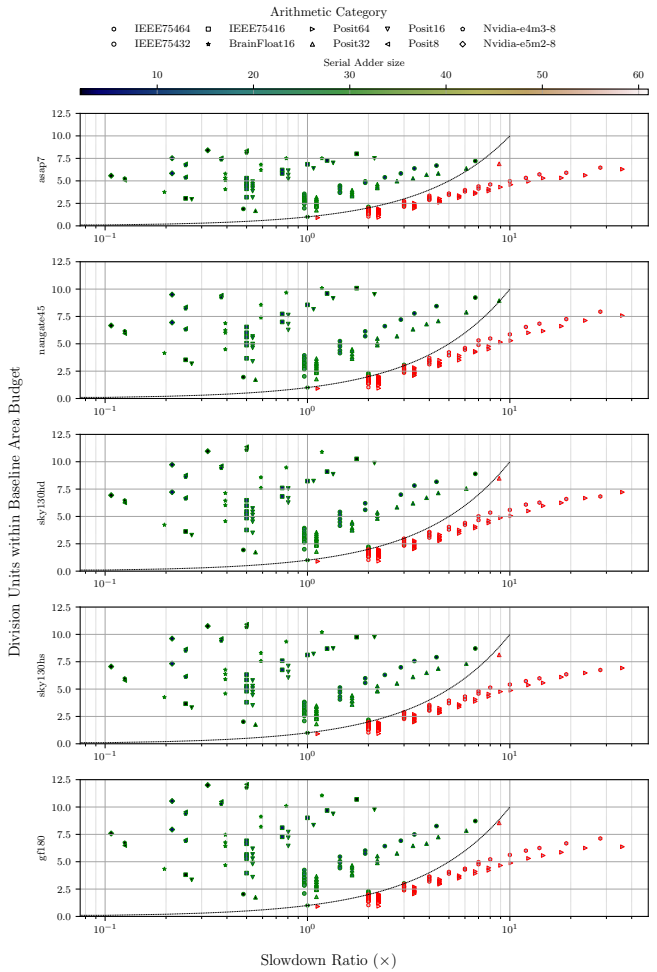
Area efficiency of SAP designs against ieee754-32 across PDKs.

SAP results against IEEE754: Area, 64-bit



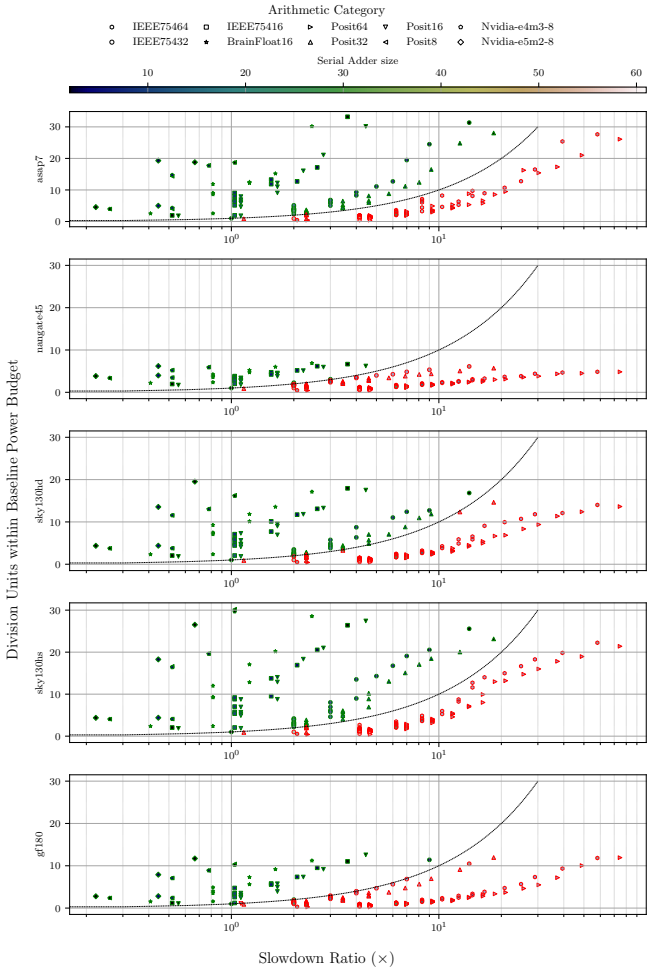
Area efficiency of SAP designs against ieee754-64 across PDKs.

SAP results against IEEE754: Area, 64-bit, log scale



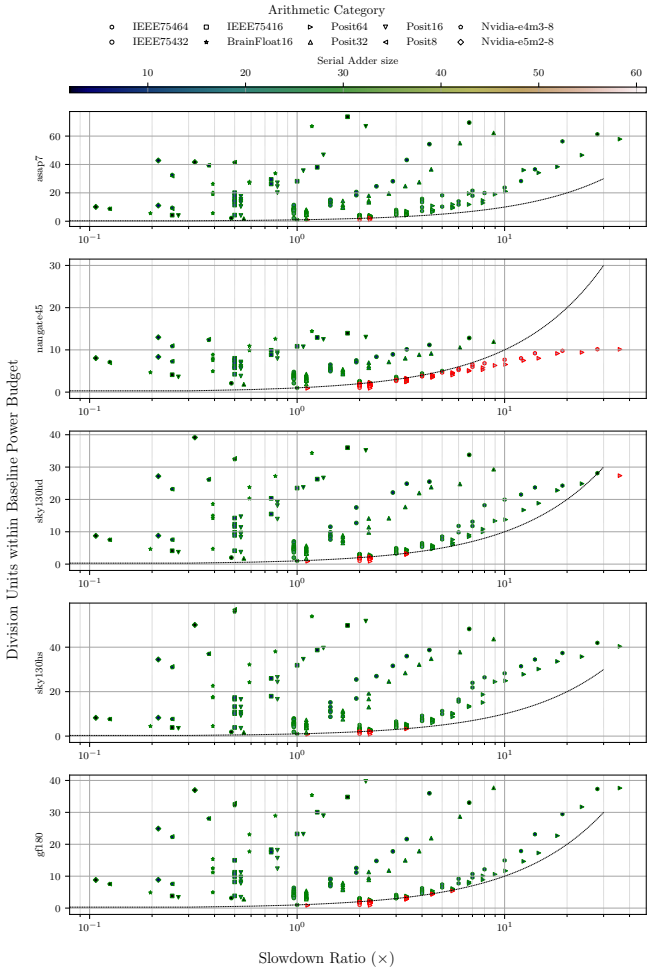
Area efficiency of SAP designs against ieee754-64 across PDKs (log scale).

SAP results against IEEE754: Power, 32-bit



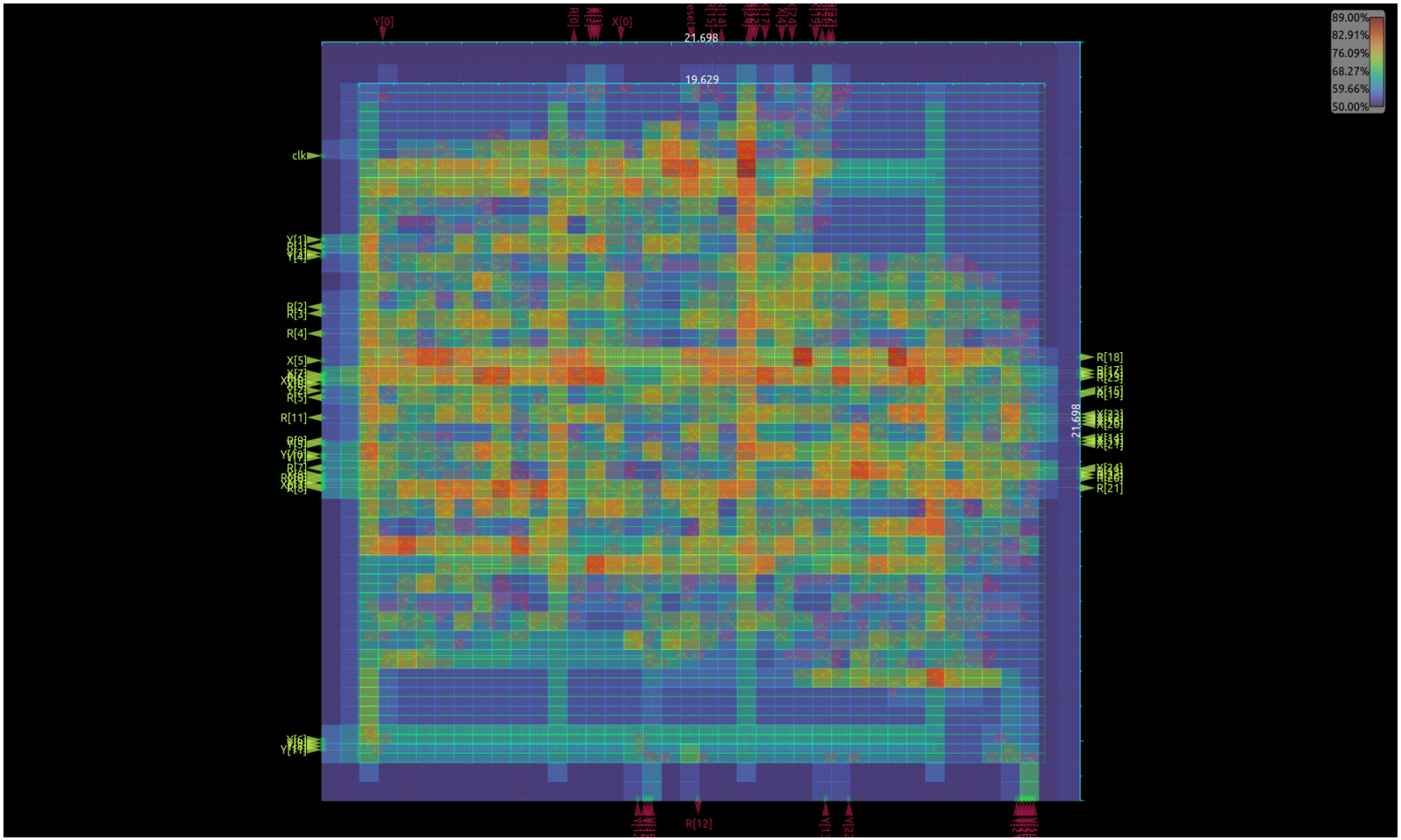
Power efficiency of SAP designs against ieee754-32 across PDKs.

SAP results against IEEE754: Power, 64-bit



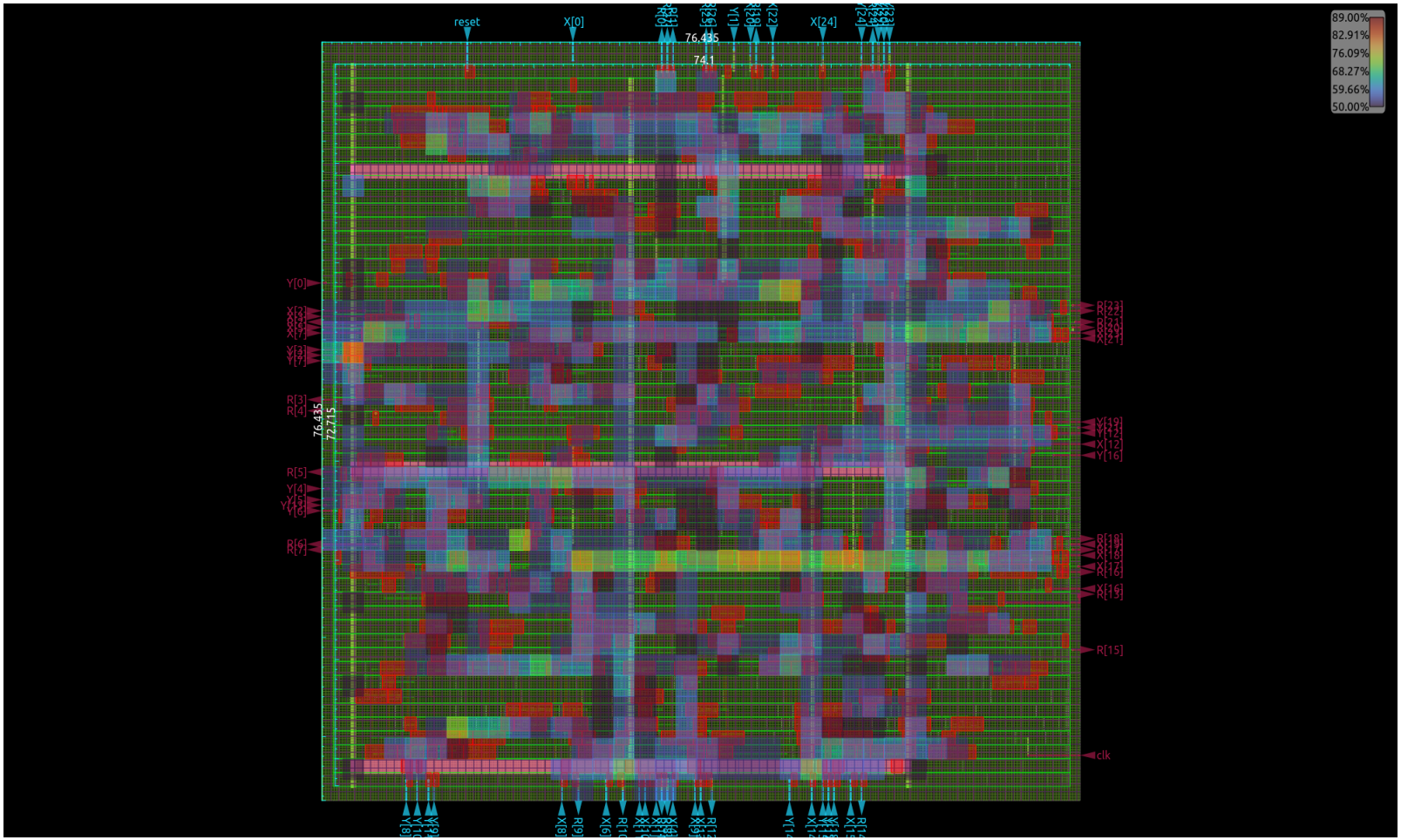
Power efficiency of SAP designs against ieee754-64 across PDKs.

GDS example: ASAP7



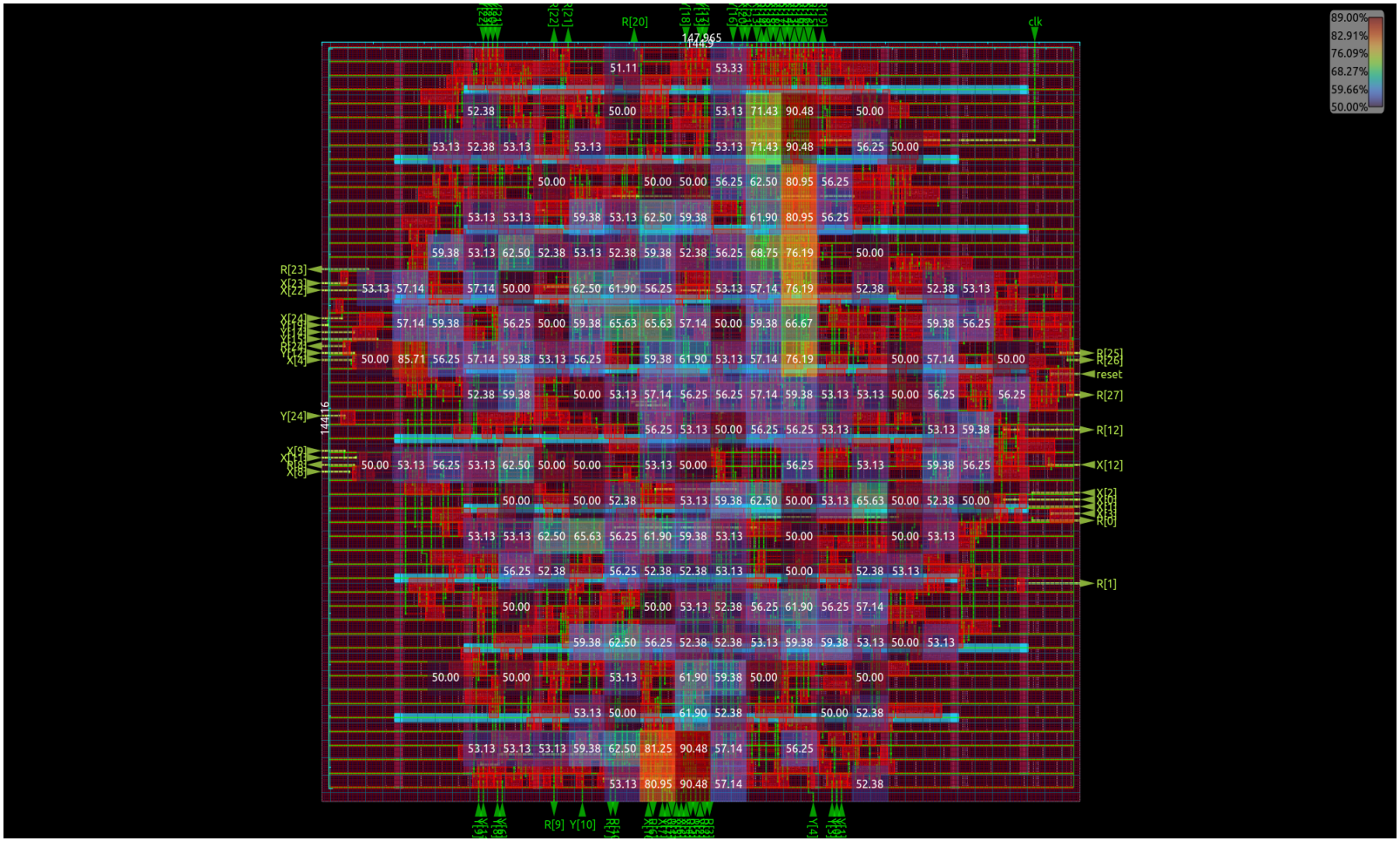
ASAP7 div. routed layout.

GDS example: NanGate45



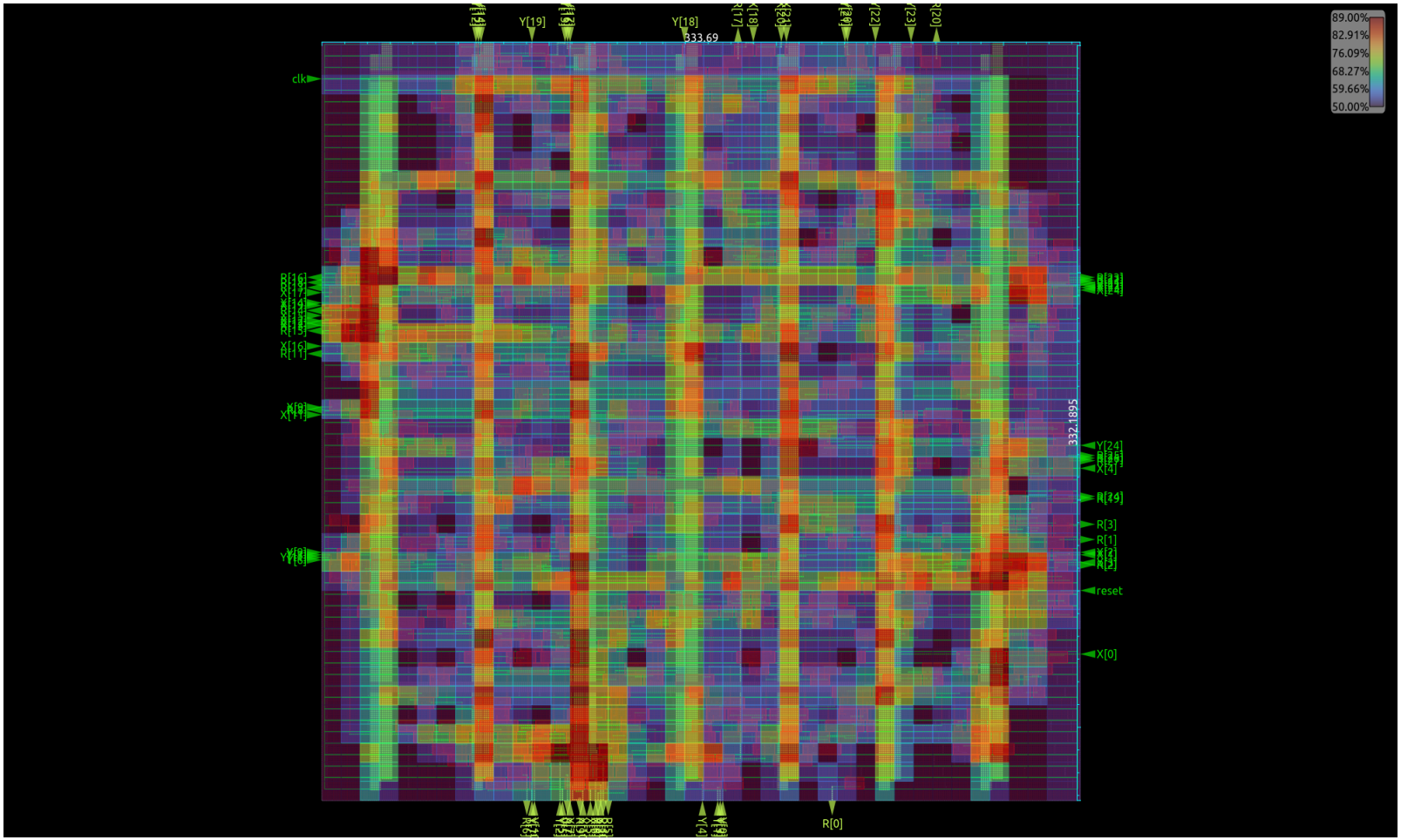
NanGate45 div. routed layout.

GDS example: Sky130HD



SkyWater130 High Density div routed layout.

GDS example: GF180



GlobalFoundries div routed layout.