The 42nd IEEE International Conference on Computer Design (ICCD 2024)
Milan, November 18-20, 2024

Extending RISC-V for Efficient Overflow Recovery in Mixed-Precision Computations

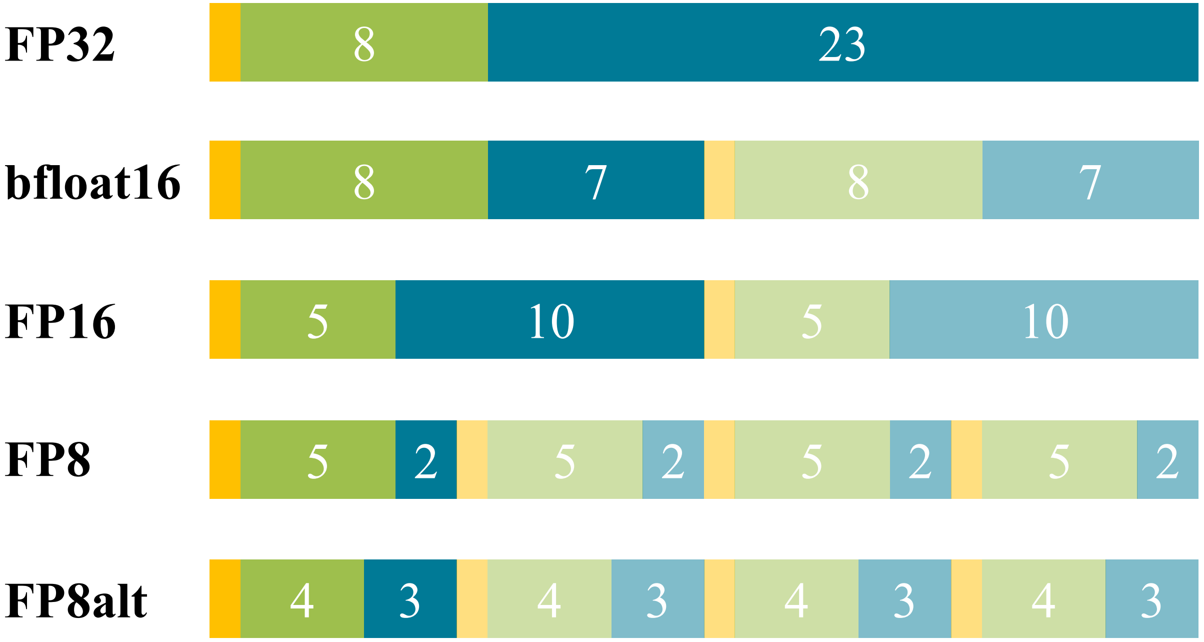
Luca Bertaccini[*], Siyuan Shen[†], Torsten Hoefler[†], Luca Benini [*‡]

[*] IIS, ETH Zurich, Switzerland,
[†] SPCL, ETH Zurich, Switzerland
[‡] DEI, University of Bologna, Italy

PULP Platform
Open Source Hardware, the way it should be!

@pulp_platform
pulp-platform.org
youtube.com/pulp_platform

Low-Precision Floating-Point Formats



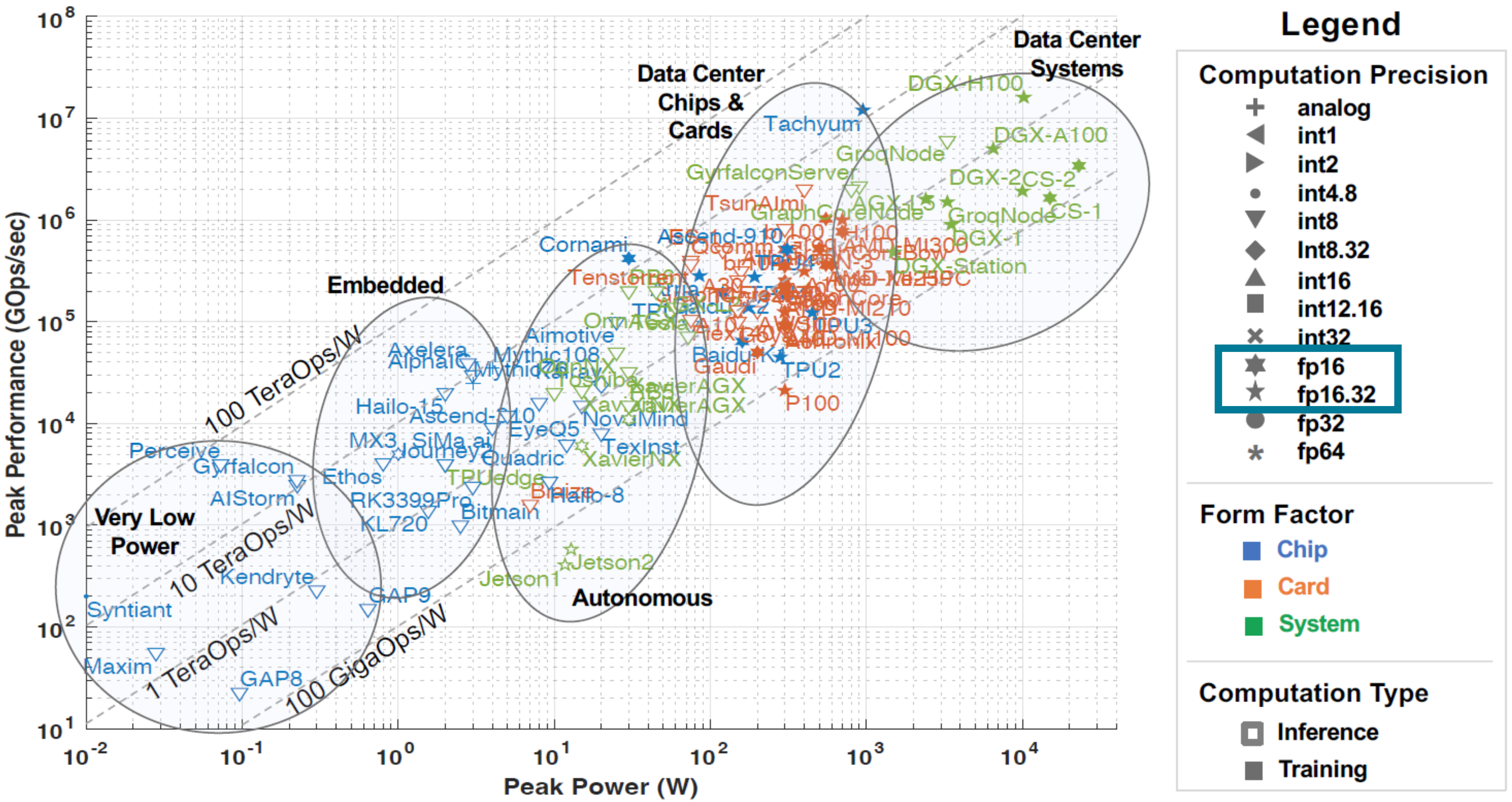
Format	# Representable values	Maximum Value
FP32	4.29 × 10^9	≈ 3.40 × 10^{38}
bfloat16	65536	≈ 3.40 × 10^{38}
FP16	65536	≈ 65504
FP8	256	≈ 49152
FP8alt	256	≈ 224

- **Low-precision** formats:
  - Higher performance and energy efficiency
  - Lower memory footprint
  - Lower data movement energy
  - Lower-accuracy results

- **Mixed-precision** operations:
  - Low-precision inputs + higher-precision accumulator
  - Low-precision benefits + retaining accuracy

Low and Mixed-Precision is Trending

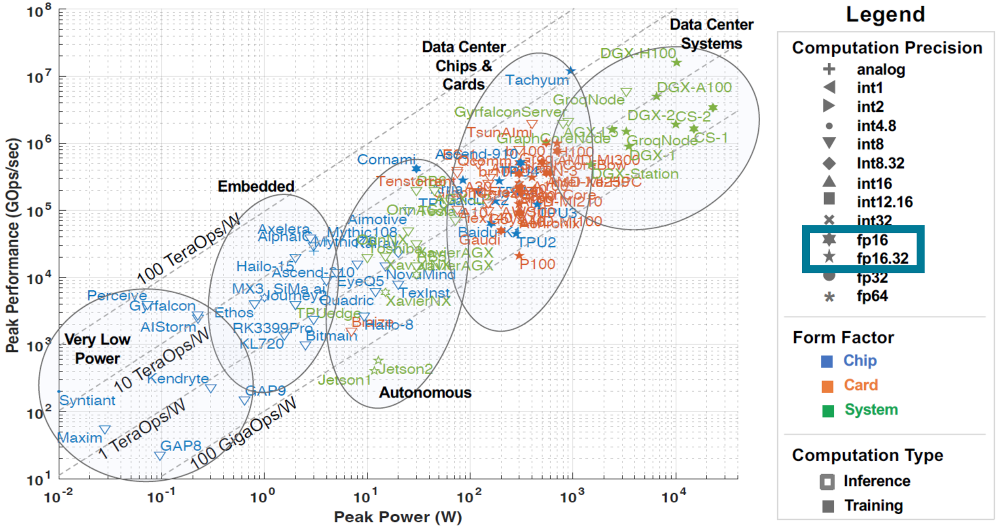
- Machine Learning
  - Many AI architectures support low and mixed-precision



A. Reuther et al., *"Lincoln AI Computing Survey (LAICS) Update"*, IEEE HPEC, 2023

Low and Mixed-Precision is Trending

- Machine Learning
  - Many AI architectures support low and mixed-precision



A. Reuther et al., *“Lincoln AI Computing Survey (LAICS) Update”,* IEEE HPEC, 2023

- More and more algorithms beyond NN being ported to low and mixed-precision [1], [2]
  - Climate modelling and weather forecast [3]
  - Audio processing [4]

- Low-precision formats are more vulnerable to overflow, which can be a destructive event

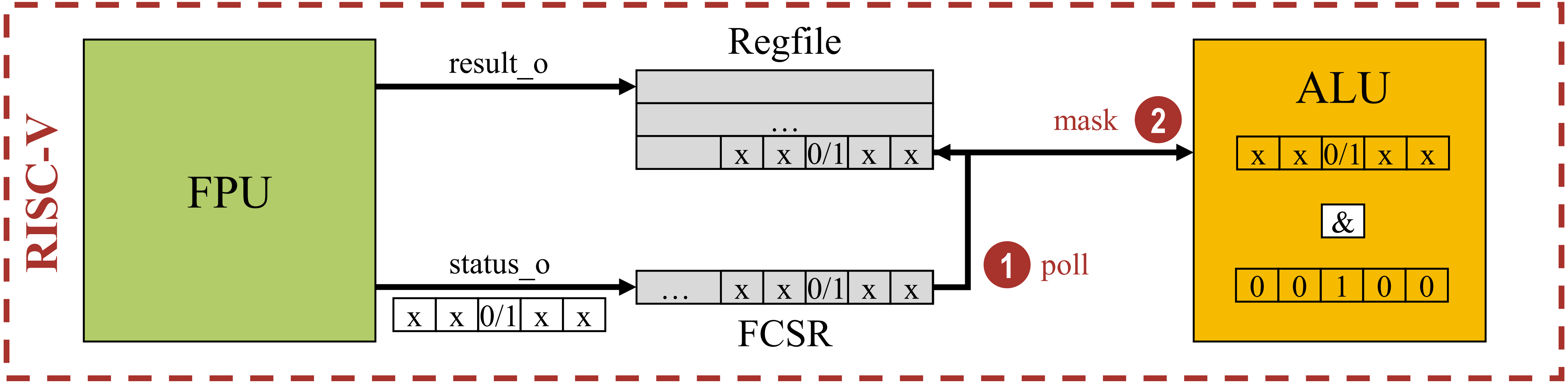
[1] N. J. Higham and T. Mary, “*Mixed precision algorithms in numerical linear algebra*”, Acta Numerica 2022

[2] M. Croci “*An overview of mixed-precision methods in scientific computing*”, 2022

[3] E. A. Paxton *et al.*, “*Climate modeling in low precision: Effects of both deterministic and stochastic rounding*”, Journal of Climate, 2022

[4] G. Cardarilli *et al.*, “*Tunable floating point for high quality audio systems: The sound of numbers*”, IEEE ACSSC, 2023

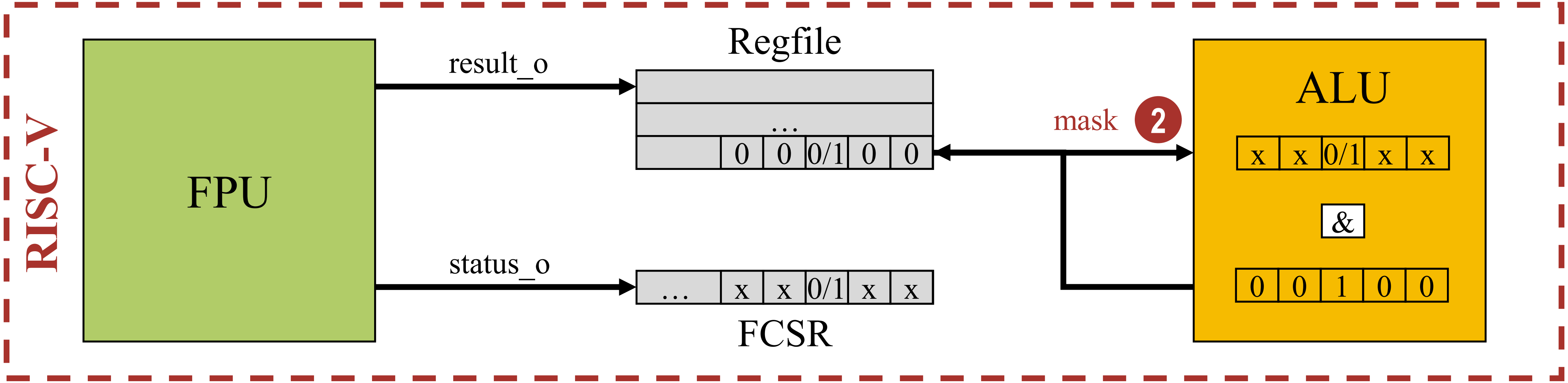
SoA Overflow Detection and Reactions



1. Detecting overflow

- Upon overflow an exception is signalled through a status flag
- Usually trapped in high-end flexible cores but not in number-crunching systems (e.g., GPUs)
- Not all ISAs natively trap FP exceptions (e.g., RISC-V)

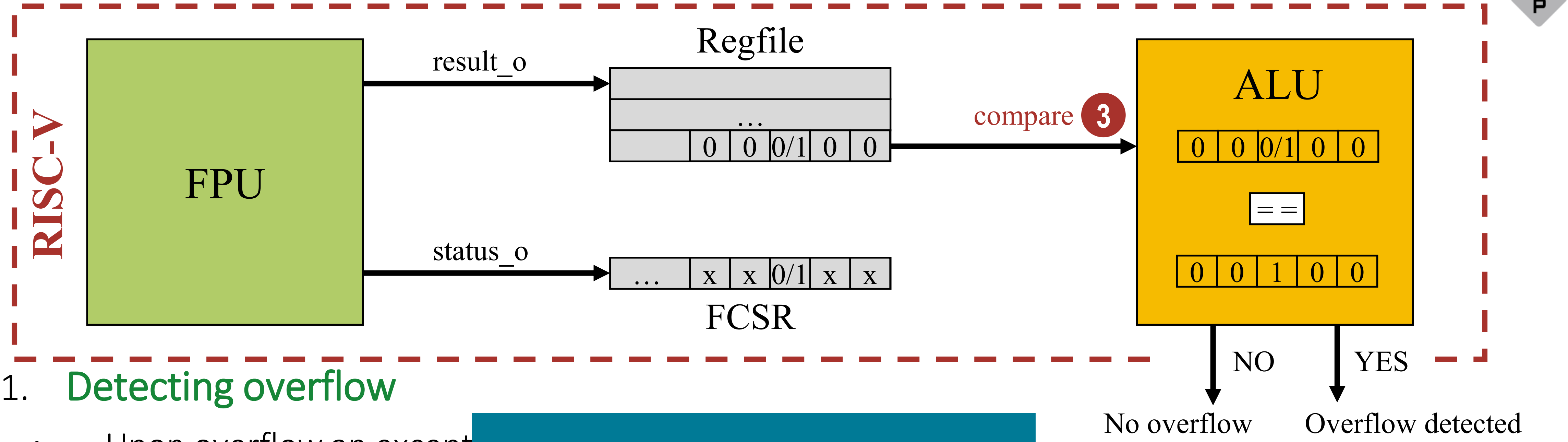
SoA Overflow Detection and Reactions



1. Detecting overflow

- Upon overflow an exception is signalled through a status flag
- Usually trapped in high-end flexible cores but not in number-crunching systems (e.g., GPUs)
- Not all ISAs natively trap FP exceptions (e.g., RISC-V)

SoA Overflow Detection and Reactions

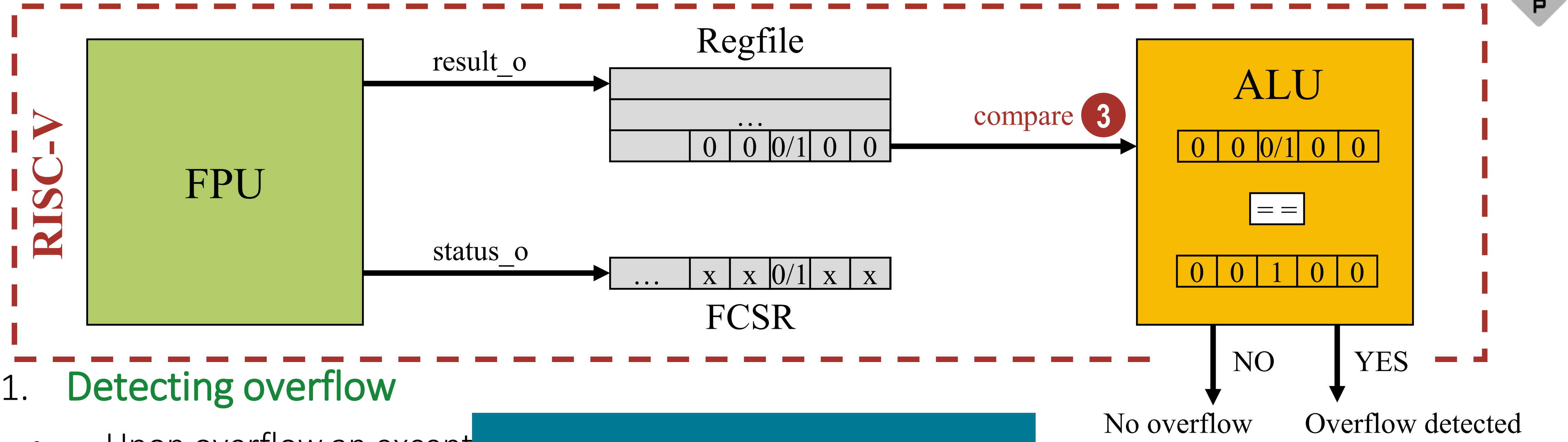


1. **Detecting overflow**
   - Upon overflow an except
   - Usually trapped in high-e unching systems (e.g., GPUs)
   - Not all ISAs natively trap FP exceptions (e.g., RISC-V)

Limiting overflow detection overhead in RISC-V systems

2. **System's reactions to overflow**
   - Producing an INF or terminating the execution
   - Handling the exception (exponent wrapping, scaling, **re-evaluating with extended range**)

SoA Overflow Detection and Reactions



1. Detecting overflow
   - Upon overflow an except
   - Usually trapped in high-e unching systems (e.g., GPUs)
   - Not all ISAs natively trap FP exceptions (e.g., RISC-V)

Limiting overflow detection overhead in RISC-V systems

2. System's reactions to overflow
   - Producing an INF or term
   - Handling the exception (e **ng with extended range**)

Leveraging mixed-precision for efficient online recovery

Contributions

1. RISC-V ISA Extension

- We extend an open-source cluster of RISC-V cores[1] to:
  - **Minimize** the **overhead** for **overflow detection**
  - **Optimize** overflow **recovery routines**

2. Efficient Overflow Recovery Routine

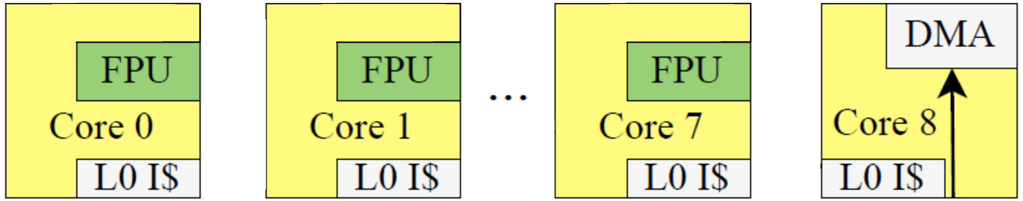
- We implement an **online recovery scheme** based on a set of **checkpoints** and leveraging a **mixed-precision ISA**, and evaluate it at an increasing probability of overflow
- Our solution adds less than 1% area overhead at a core level and we show that it can be tuned to recover from a single overflow in a 128x128 matmul at only 3% performance penalty

[1] https://github.com/pulp-platform/snitch_cluster

Baseline System

MiniFloat Snitch Cluster of RISC-V Cores[1]

- 8 RISC-V cores coupled with 64-bit multi-format SIMD FPUs + 1 DMA core

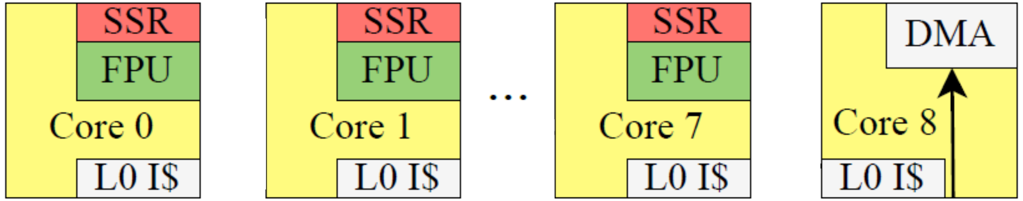


[1] https://github.com/pulp-platform/snitch_cluster

Baseline System

MiniFloat Snitch Cluster of RISC-V Cores[1]

- 8 RISC-V cores coupled with 64-bit multi-format SIMD FPUs + 1 DMA core
- Enhanced with streaming ISA extensions + hardware-managed loops
- Up to more than 90% of FPU utilization (i.e., a new FPU instruction issued in 90% of the cycles)

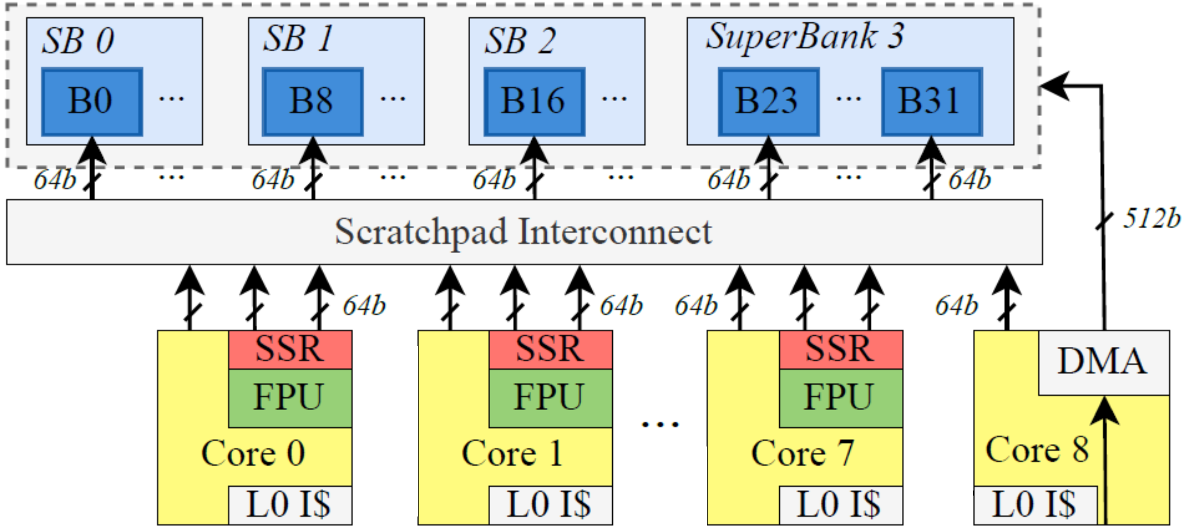


[1] https://github.com/pulp-platform/snitch_cluster

Baseline System

MiniFloat Snitch Cluster of RISC-V Cores[1]

- 8 RISC-V cores coupled with 64-bit multi-format SIMD FPUs + 1 DMA core
- Enhanced with streaming ISA extensions + hardware-managed loops
- Up to more than 90% of FPU utilization (i.e., a new FPU instruction issued in 90% of the cycles)
- 32 banks of scratchpad memory

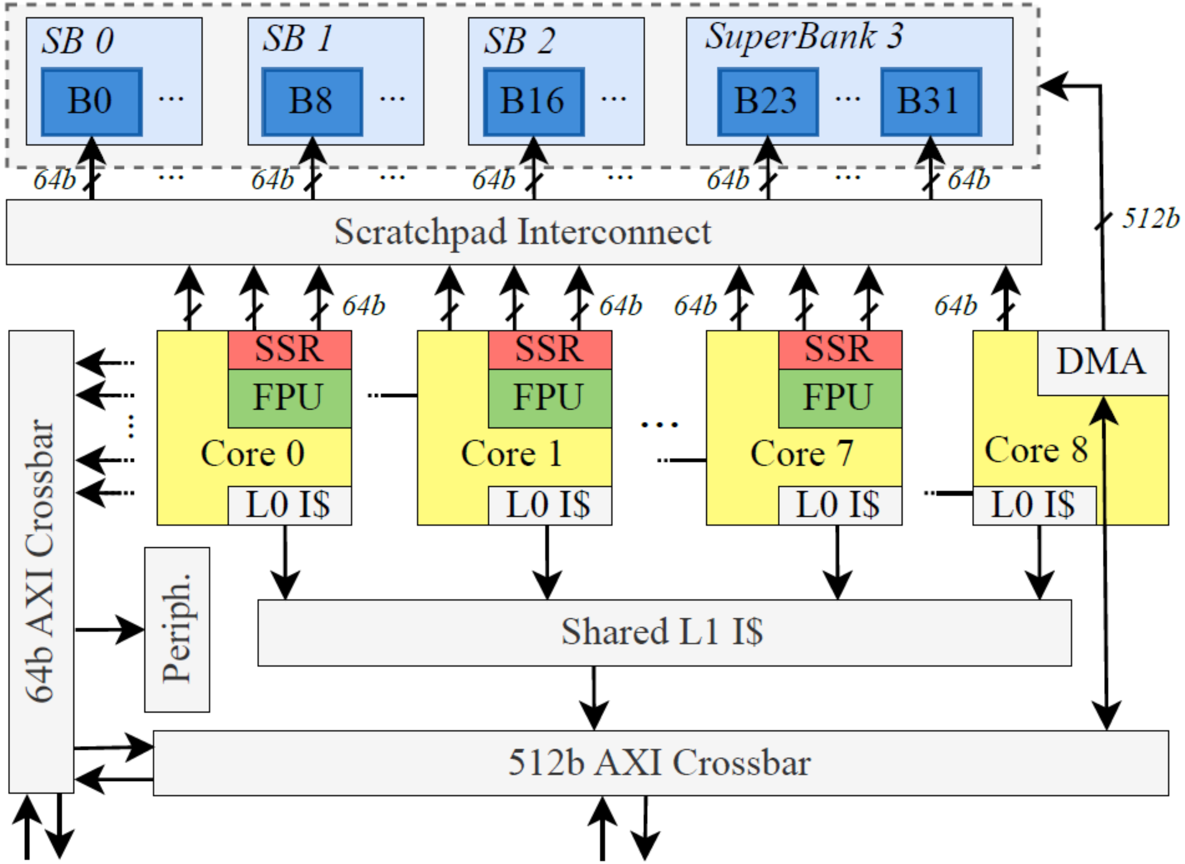


[1]https://github.com/pulp-platform/snitch_cluster

Baseline System

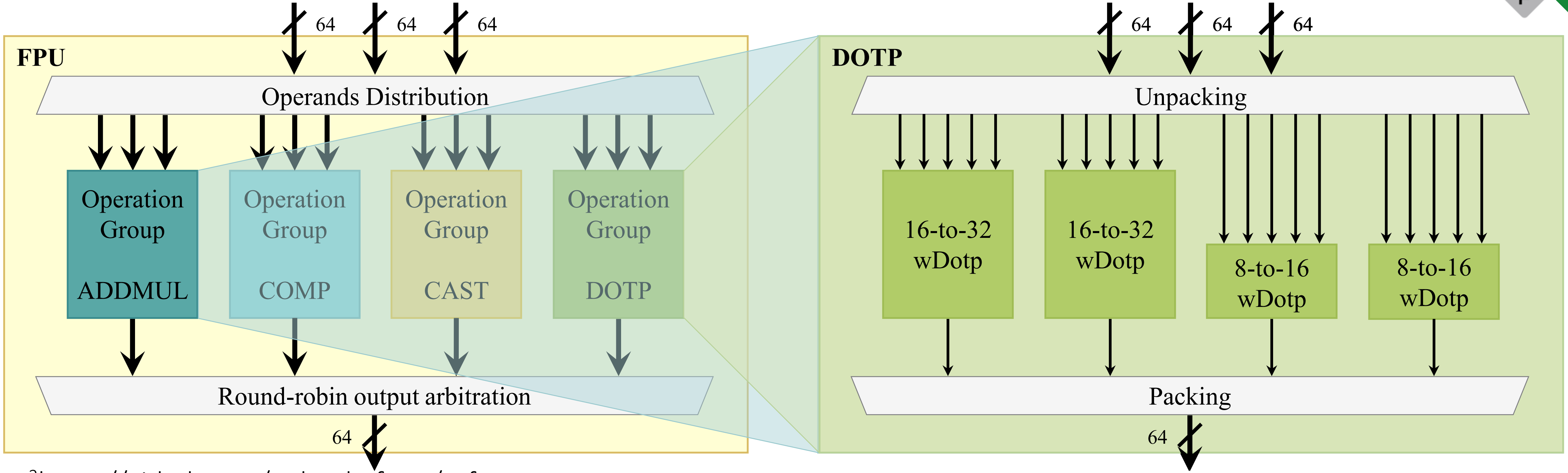
MiniFloat Snitch Cluster of RISC-V Cores[1]

- 8 RISC-V cores coupled with 64-bit multi-format SIMD FPUs + 1 DMA core
- Enhanced with streaming ISA extensions + hardware-managed loops
- Up to more than 90% of FPU utilization (i.e., a new FPU instruction issued in 90% of the cycles)
- 32 banks of scratchpad memory



[1] https://github.com/pulp-platform/snitch_cluster

Baseline System’s FPUs



[2]https://github.com/pulp-platform/cvfpu

Multi and Mixed-Precision CVFPU[2]

- Support for 8/16-bit FP dot-product instructions with accumulation in 16/32-bit FP
- Two 8-to-16b wDotp units + Two 16-to-32b wDotp units
- Four 8-to-16b wDotp operations or two 16-to-32b wDotp operations

Higher-Precision Accumulation in SoA Architectures

- **FP8 with FP16 accumulation (FP16 max ≈ 65k, vulnerable to overflow)**
  - Snitch
  - IBM AI Chip (S. K. Lee *et al.*, "*A 7-nm four-core mixed-precision AI chip with 26.2-TFLOPS hybrid-FP8 training, 104.9-TOPS INT4 inference, and workload-aware throttling*", IEEE JSSC 2021)
  - NVIDIA H100

- **FP8 with FP32 accumulation (higher cost)**
  - Tesla Dojo
  - Intel Gaudi 3
  - Previous SoA studies → Higher cost for FP32 accumulation (M. van Baalen *et al.*, "*FP8 versus INT8 for efficient deep learning inference*", 2023)

- **FP8 with FP16 + FP32 accumulation**
  - NVIDIA ADA using H100 supports both FP16 and FP32 accumulation
  - Half the performance when accumulating in FP32

Higher-Precision Accumulation in SoA Architectures

- **FP8 with FP16 accumulation (FP16 max ≈ 65k, vulnerable to overflow)**
  - Snitch
  - IBM AI Chip (S. K. Lee *et al.*, “*A 7-nm four-core mixed-precision AI* *ce, and workload-aware throttling*”, IEEE JSSC 2021)
  - NVIDIA H100

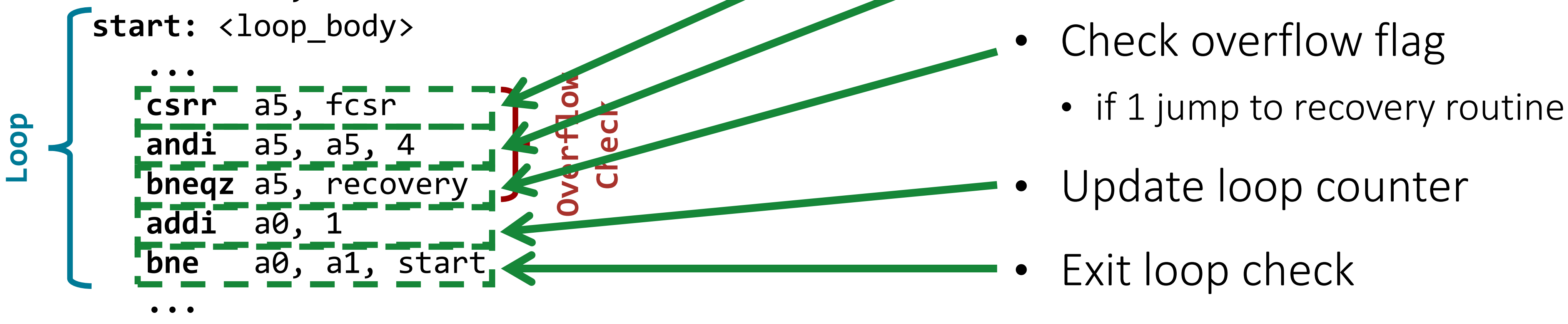
- We apply a similar approach to support 8-to-32b wDotp on Snitch
- Reusing the modules already available (16-to-32b wDotp units)

- **FP8 with FP32 accumulation (higher cost)**
  - Tesla Dojo
  - Intel Gaudi 3
  - Previous SoA studies → Higher cost for FP32 accumulation (M. van Baalen *et al.*, “*FP8 versus INT8 for efficient deep learning inference*”, 2023)

- **FP8 with FP16 + FP32 accumulation**
  - NVIDIA ADA using H100 supports both FP16 and FP32 accumulation
  - Half the performance when accumulating in FP32

Overflow-Conditioned Branches for Low-Cost Detection

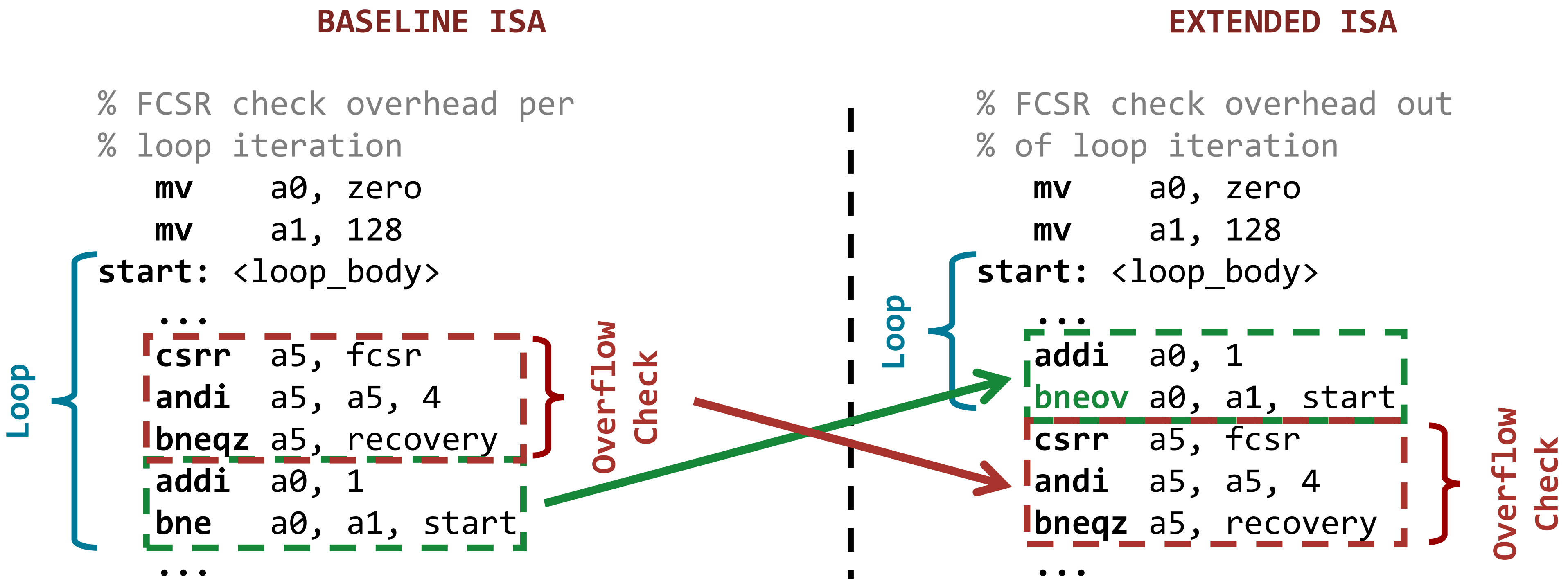
BASELINE ISA



```
% FCSR check overhead per
% loop iteration
   mv    a0, zero
   mv    a1, 128
start: <loop_body>
   ...
   csrr  a5, fcsr
   andi  a5, a5, 4
   bneqz a5, recovery
   addi  a0, 1
   bne   a0, a1, start
   ...
```

- Poll FCSR
- Mask overflow flag
- Check overflow flag
  - if 1 jump to recovery routine
- Update loop counter
- Exit loop check

Overflow-Conditioned Branches for Low-Cost Detection



BASELINE ISA

```
% FCSR check overhead per
% loop iteration
   mv     a0, zero
   mv     a1, 128
start: <loop_body>
   ...
   csrr   a5, fcsr
   andi   a5, a5, 4
   bneqz  a5, recovery
   addi   a0, 1
   bne    a0, a1, start
   ...
```

EXTENDED ISA

```
% FCSR check overhead out
% of loop iteration
   mv     a0, zero
   mv     a1, 128
start: <loop_body>
   ...
   addi   a0, 1
   bneov  a0, a1, start
   csrr   a5, fcsr
   andi   a5, a5, 4
   bneqz  a5, recovery
   ...
```

- Overflow-conditioned branch instructions (**bneov**) to move the overflow check outside the loop
- Exiting the loop either if the loop completed its iterations or an overflow has been raised
- Outside the loop → overflow check to detect whether the loop has been exited because of overflow

Mixed Precision for Efficient Overflow Recovery

Loop
```
% 8-to-16b wDotp
wDotp.h.b  fa0, ft0, ft1
```

Recovery

BASELINE ISA

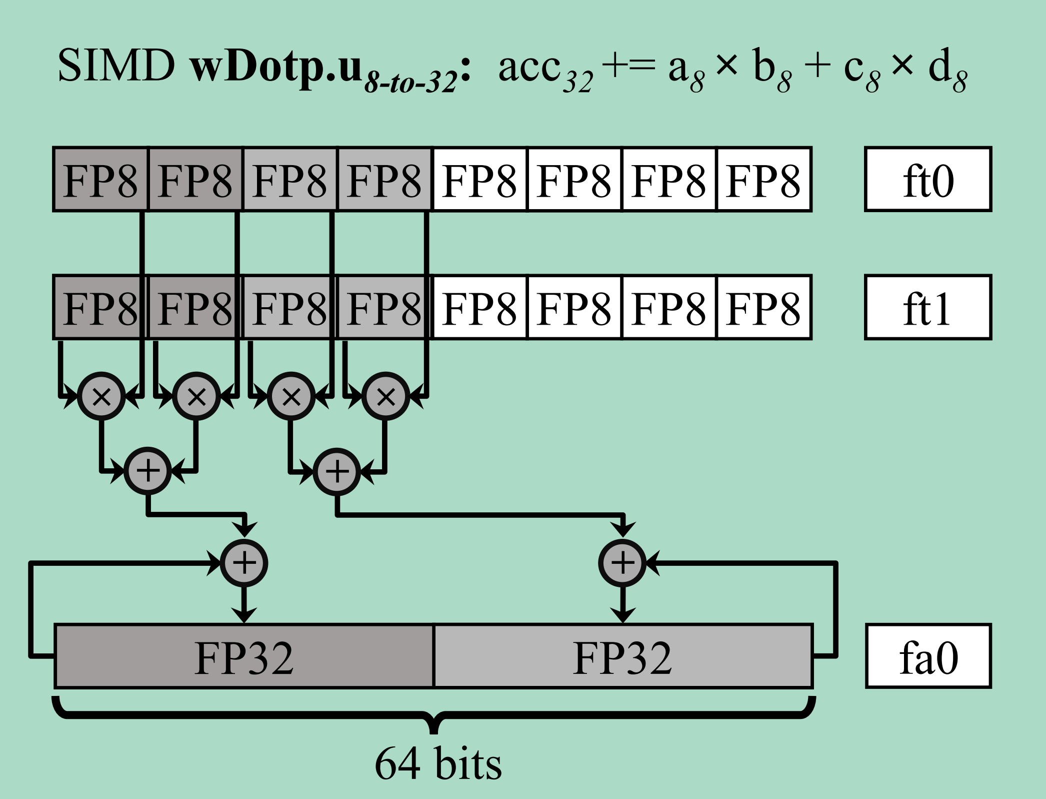
Loop
```
% 8-to-16-to-32b wDotp
fcvtl.h.b  fa2, ft0
fcvtu.h.b  fa3, ft0
fcvtl.h.b  fa4, ft1
fcvtu.h.b  fa5, ft1
wDotp.s.h  fa0, fa2, fa4
wDotp.s.h  fa1, fa3, fa5
```

EXTENDED ISA

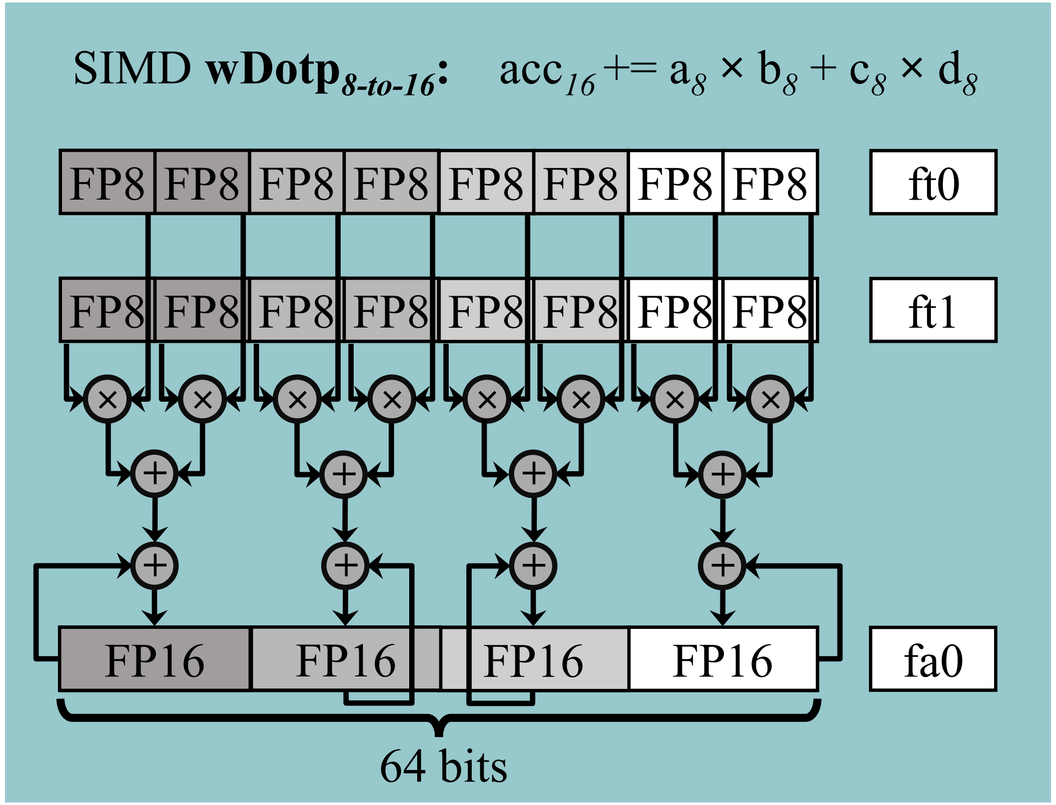
Loop
```
% 8-to-32b wDotp
wDotpl.s.b  fa0, ft0, ft1
wDotpu.s.b  fa1, ft0, ft1
```

- Online recovery on baseline ISA requires converting the inputs (8b->16b) to higher-precision and then compute with a higher-precision accumulation (16-to-32-b widening dot product)
- ISA extension adding two 8-to-32-bit widening dot-products to limit the overhead

Mixed Precision Widening Operations



SIMD **wDotp.u**$_{8\text{-}to\text{-}32}$: acc_{32} += $a_8 \times b_8 + c_8 \times d_8$

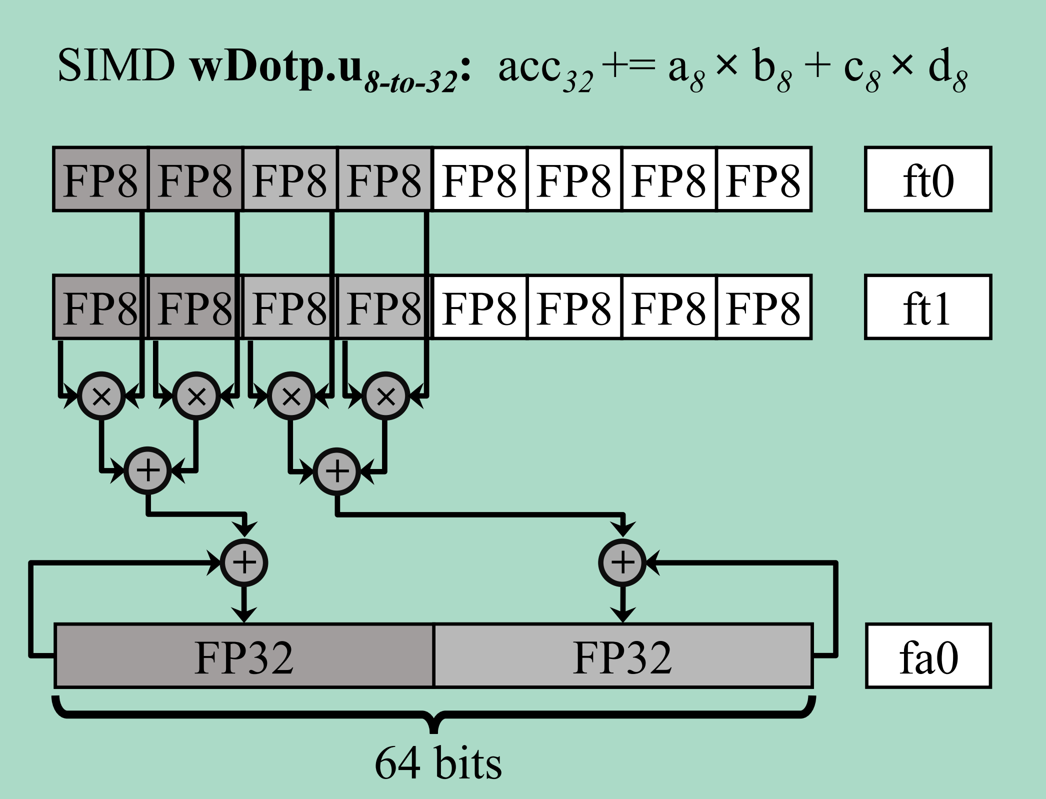


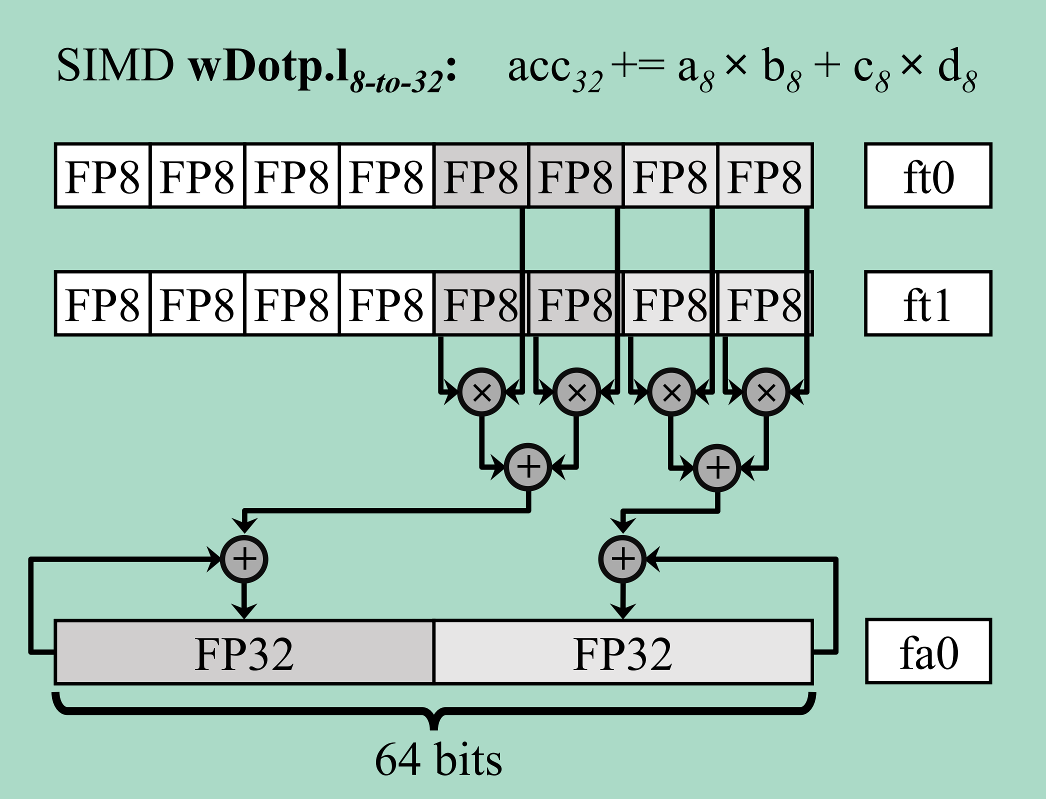
SIMD **wDotp**$_{8\text{-}to\text{-}16}$: acc_{16} += $a_8 \times b_8 + c_8 \times d_8$

- 8-to-32-bit wDotp
- 8 FLOP/cycle/FPU
- Computed on the two 16-to-32 wDotp units

- 8-to-16-bit wDotp
- 16 FLOP/cycle/FPU

Mixed Precision Widening Operations


SIMD **wDotp.u**$_{8\text{-}to\text{-}32}$: $acc_{32} += a_8 \times b_8 + c_8 \times d_8$
FP8 FP8 FP8 FP8 FP8 FP8 FP8 FP8
ft0
FP8 FP8 FP8 FP8 FP8 FP8 FP8 FP8
ft1
FP32 FP32
fa0
64 bits


SIMD **wDotp.l**$_{8\text{-}to\text{-}32}$: $acc_{32} += a_8 \times b_8 + c_8 \times d_8$
FP8 FP8 FP8 FP8 FP8 FP8 FP8 FP8
ft0
FP8 FP8 FP8 FP8 FP8 FP8 FP8 FP8
ft1
FP32 FP32
fa0
64 bits

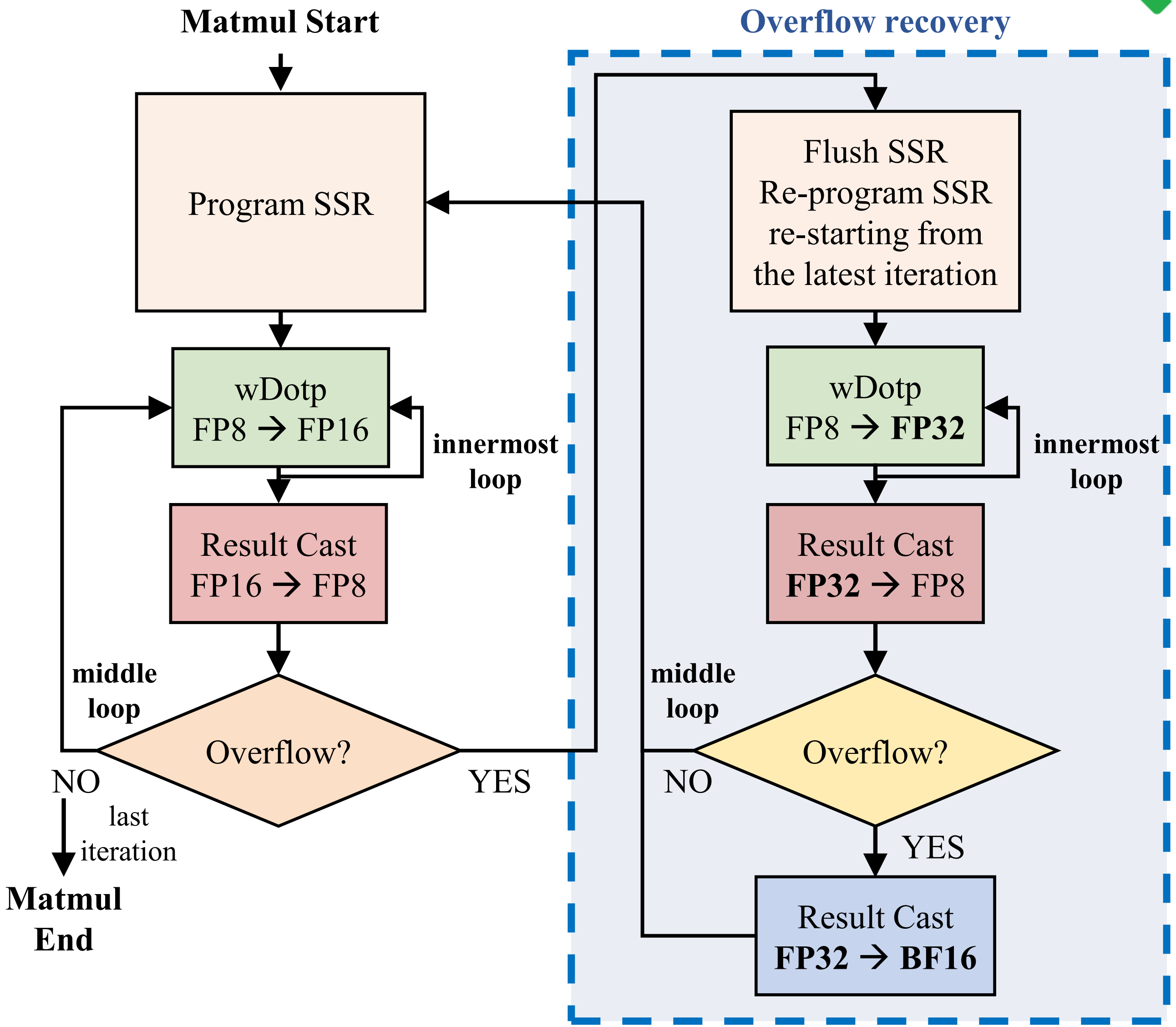
- 8-to-32-bit wDotp
- 8 FLOP/cycle/FPU
- Computed on the two 16-to-32 wDotp units

- Two instructions to process the upper and lower parts of the inputs

Overflow Recovery Routine

- Based on a set of **checkpoints**

- Overflow recovery routine:
  - **Rolls back**
  - **Re-evaluate** at **higher precision** the portion between two checkpoints
    - 8-to-32b wDotp
    - Half the performance during the re-evaluation
  - **Resume** normal operation

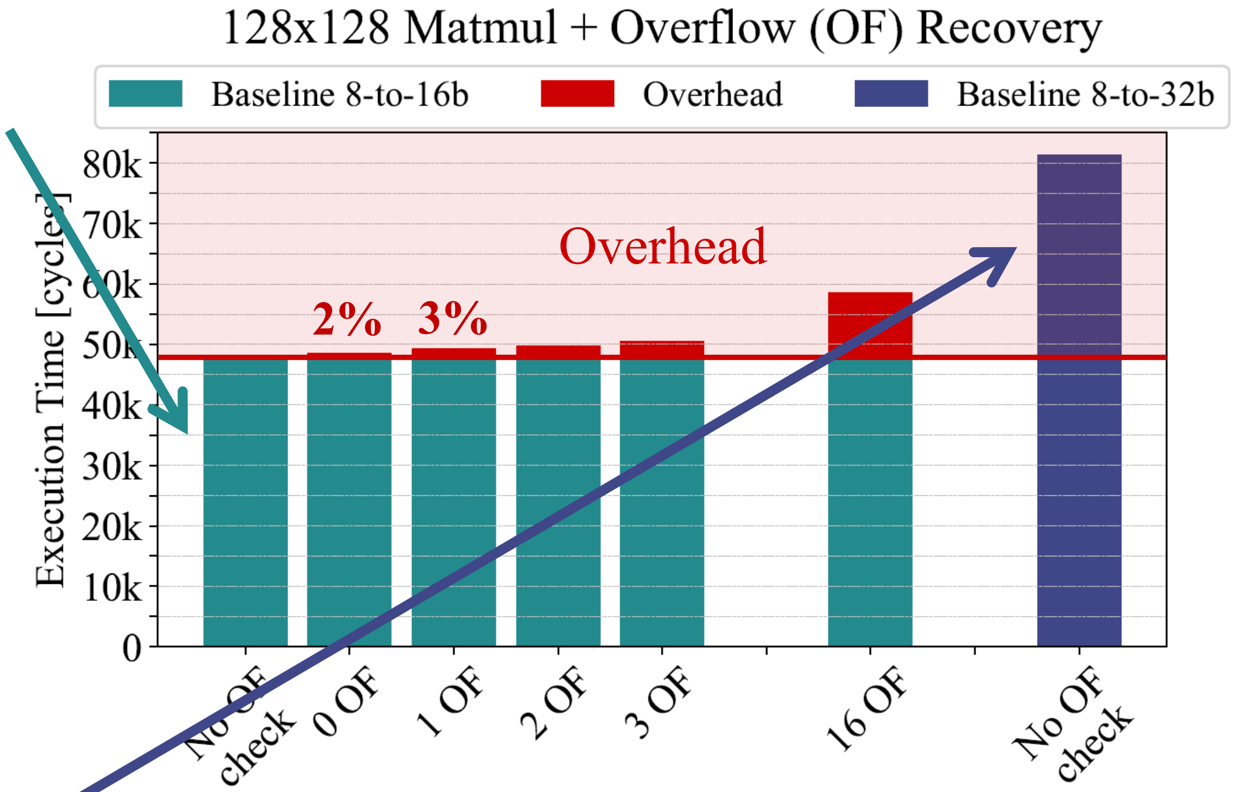


Results

- Implemented the extended MiniFloat Snitch cluster with Synopsys Fusion Compiler in a 12nm technology

Fragile baseline

- Less than 1% of area overhead to Snitch core
- On a 128x128 matmul, tuned the checkpoints to achieve:
  - 2% performance penalty in the absence of overflow (overflow detection overhead)
  - 3% performance penalty to detect and recover one overflowing portion of code
- Computing the whole kernel with higher precision accumulation would be 1.7x slower



Conclusions

- Extended a cluster of streaming RISC-V cores supporting low and mixed-precision operations for efficient overflow recovery at less than 1% area overhead at a RISC-V core level
- Devised an overflow recovery routine
- Showed that the checkpoints can be tuned to pay 3% of performance for recovery one overflow which would otherwise be destructive
- 1.7x faster than computing always at higher precision

Thank you!

@pulp_platform

pulp-platform.org

youtube.com/pulp_platform