

EPI Forum

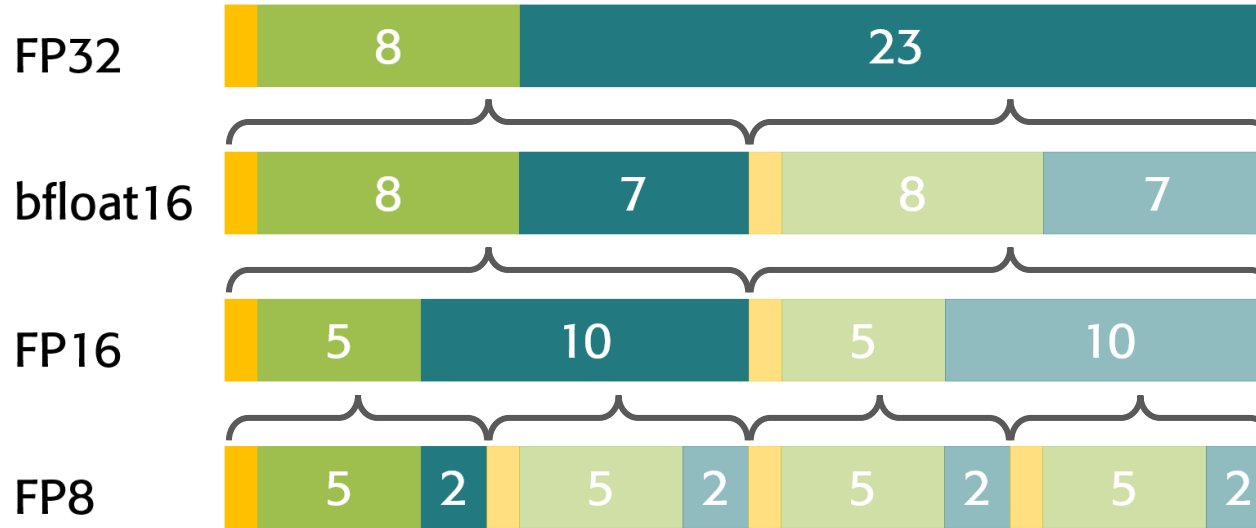
Barcelona, 9-10.10.2024.





-
- The diagram illustrates the Snitch RV32I Core architecture. The core is a gray box containing several components: a Score-board, Decode, Regfile, LSU, and ALU. It is connected to memory via Shared L1 I\$ and LO I\$ blocks. The core also has an Acc. Port connected to the FPU Subsystem (yellow box), which includes a FREP Sequencer, FPU, and SSRs. Green arrows indicate data flow between memory and the core/FPU.

Low & Mixed-Precision Floating-Point Formats



Format	# Representable values	Maximum Value
FP32	4.29×10^9	$\approx 3.40 \times 10^{38}$
Bfloat16	65536	$\approx 3.40 \times 10^{38}$
FP16	65536	≈ 65504
FP8	256	≈ 57344

- **Low precision**

- Lower memory footprint
- Higher energy efficiency
- Higher throughput with SIMD

- **Mixed Precision**

- Addresses accuracy loss of low precision format
- Accumulation in higher precision format
 - e.g. **wDotp** FP8->FP16

SSR & FREP: the Key for PE efficiency



- **SSR**: Link register read/writes into implicit LD/ST
 - Extension around the core's register file
 - Address generators (2-3KGE/SSR)
 - Configured out of inner loop (LD/ST elision)
 - Staggering: generators prefetch from memory (latency tolerant!)
- **FREP**: L0 instruction buffer (no I\$ access)
 - Pseudo-dual issue (Int pipeline can proceed in parallel)
 - No boundary checking for loop (similar HW loop in DSPs)
- **Boost FPU utilization → 100% (once setup is amortized)**

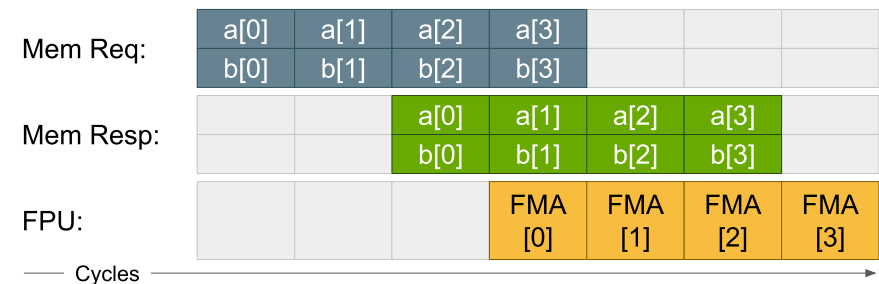
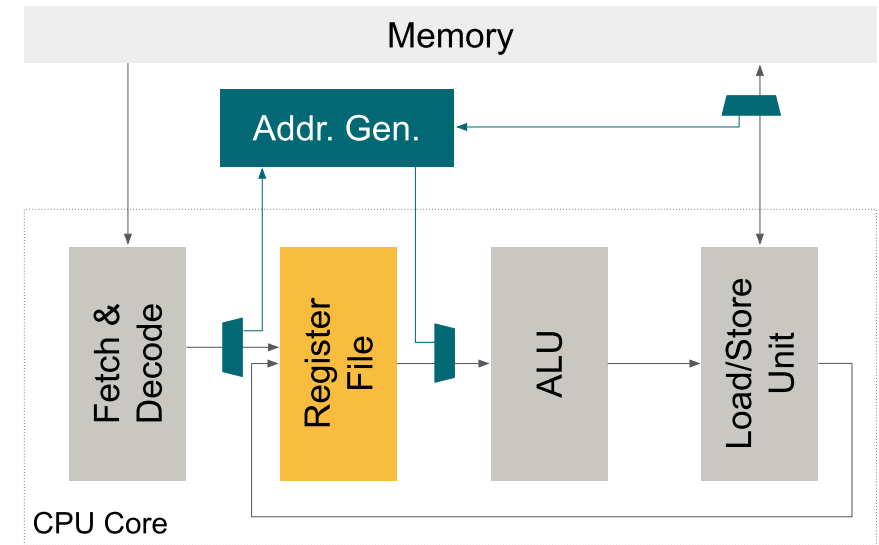
dotp: 30% FPU

```
loop:
fld r0, %[a]
fld r1, %[b]
fmadd r2, r0, r1
```



dotp: 90% FPU

```
scfg 0, %[a], ldA
scfg 1, %[b], ldB
loop:
fmadd r2, ssr0, ssr1
```

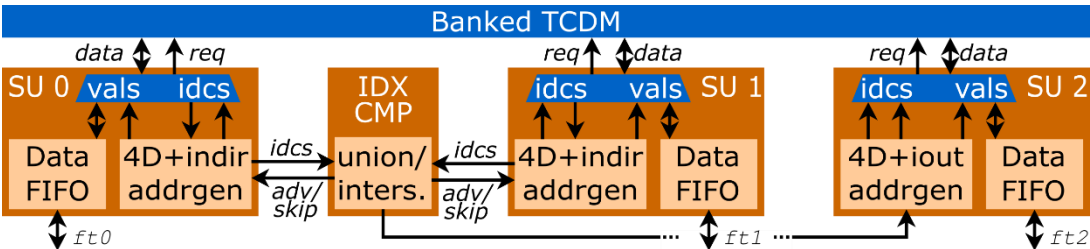


Less expensive than OoO (CPU) and Multi-threading (GPU)

Accelerating Dense, Sparse, and Mixed Workloads



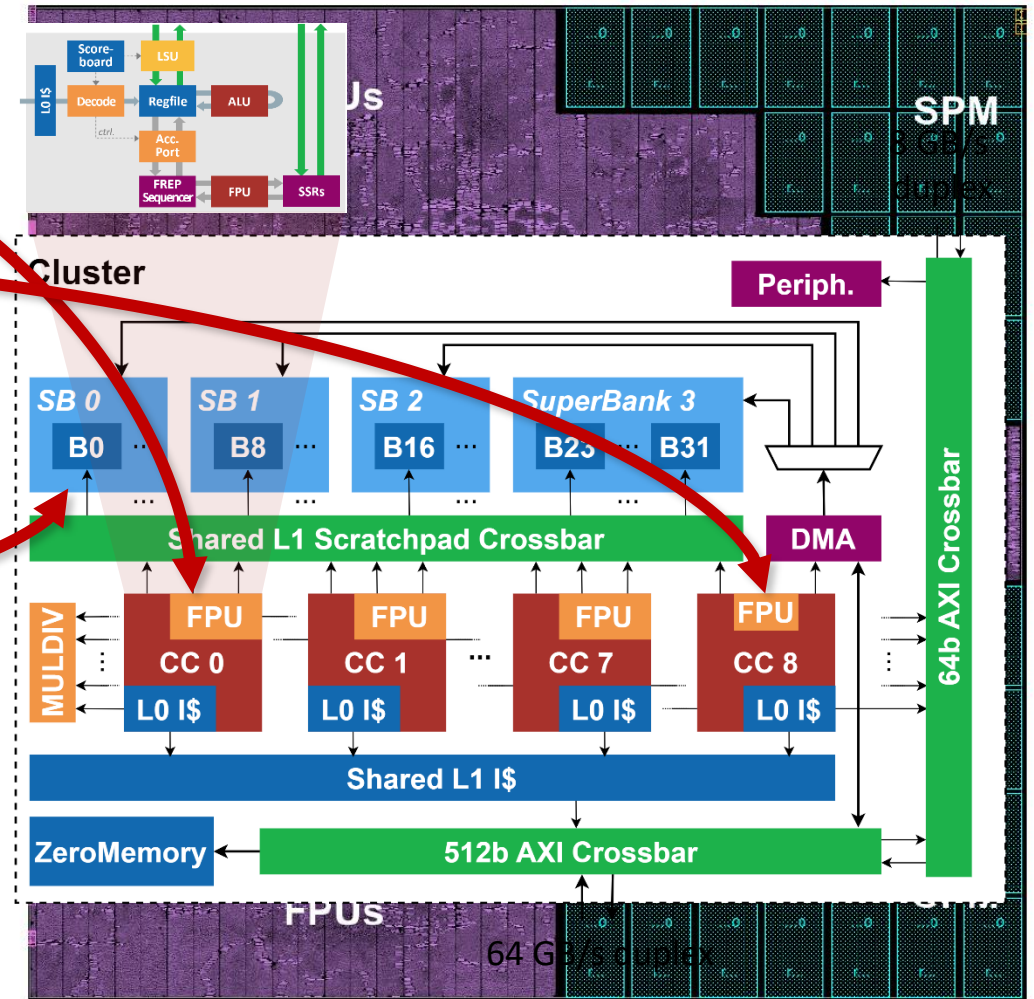
- **Sparse SSRs (SSSRs) are highly flexible**
 - 3 *affine* streams for *dense* workloads
 - 2 *indirect* streams for *scatter-gather* (sparse-dense LA, stencils, codebooks...)
 - *Intersection & union* for *sparse-sparse* LA
- **Accelerate *mixed* compute patterns underlying emergent HPC & ML**
 - Often *combine* sparse and dense compute → must handle both efficiently
 - HPC: sparse solvers and stencil codes
 - ML: sparsified *weights* and *activations*, unconstrained *quantization* w. codebooks



Workload (✓: essential, +: beneficial)		dd. LA	sd. LA	ss. LA	stencils	codebk
HPC	Conjugate Gradient	✓	✓			
	Finite difference methods	✓			✓	
	ADMM MPC	✓	✓			+
ML	Convolutional NNs	✓	+			+
	Deep NNs	✓	+	+		+
	Graph NNs	✓	✓	+		+
	Large Language Models	✓	+			+

STX Cluster

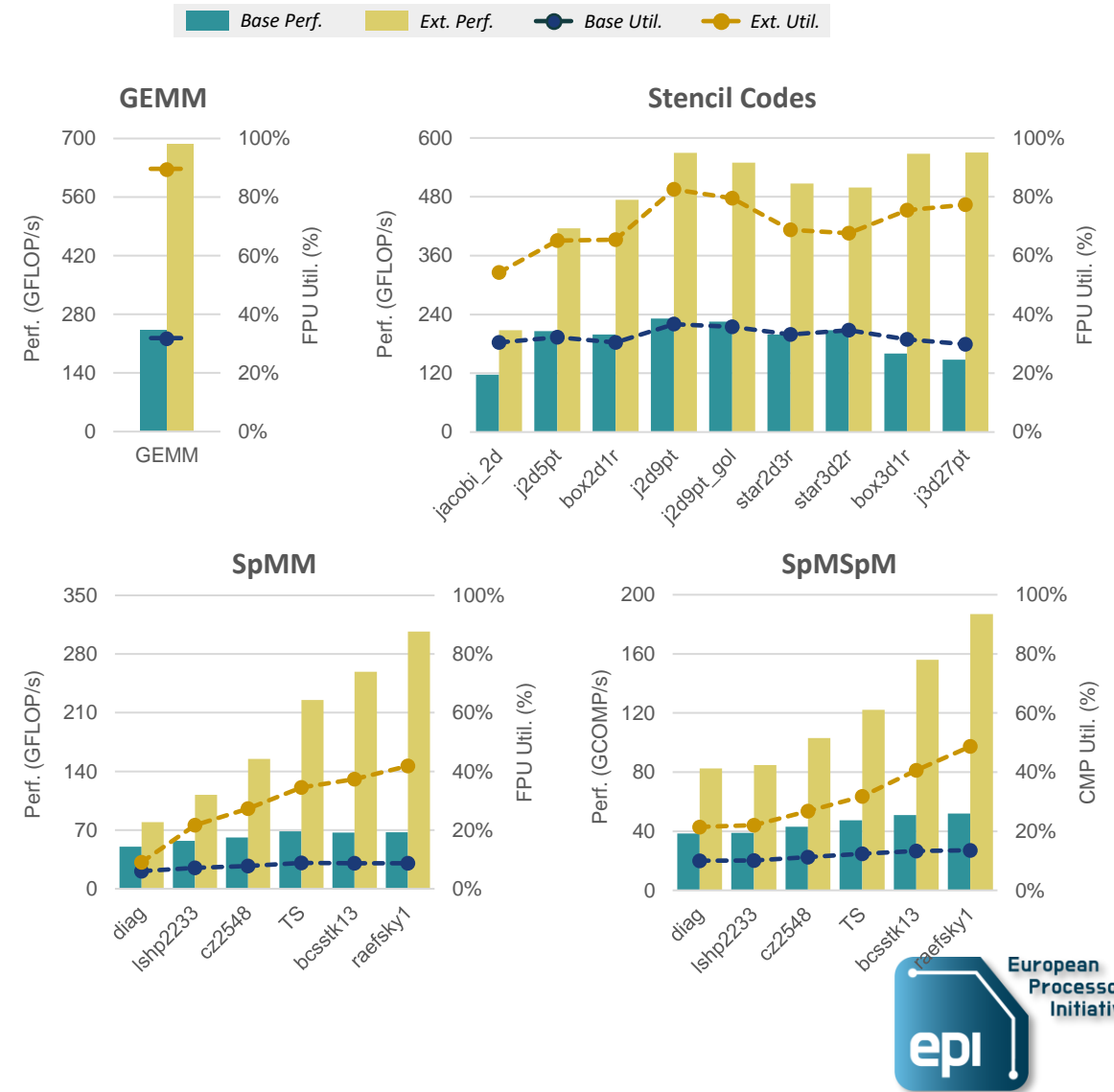
- **8 Snitch compute cores**
 - SIMD 64b FPU with SSRs & FREP
- **9th Core: DMA engine**
 - 512b interface to interconnect
 - HW support for autonomous $\leq 2D$ transfers, higher dimensions through SW
 - *Latency-tolerance block transfers (100s of cycles)*
- **128 KiB TCDM**
 - 32-bank, low-latency shared scratchpad
 - Double-buffer large chunks with DMA
- **Shared TDCDM, I-cache and peripherals**
- **Shared DMA (10% overhead) for global latency tolerance**



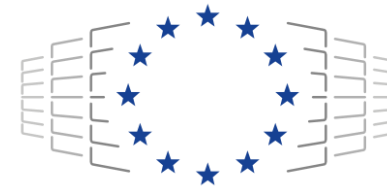
STX Cluster Performance and FPU Utilization



- **Mixed workloads @1GHz, 0.8V, 25°C**
 - Occamy-chip, 2x24-cluster system
 - **RV32G baseline** vs. **code using ISA extensions**
 - All workloads use FP64 data, int16 indices
 - Sparse LHS real-world matrices, RHS 1% density
- **Near-ideal dense, leading sparse perf.**
 - **GEMM**: **14.3 GFLOP/s**, **2.8×** faster than RV32G, **89%**FPU util. competitive with GPUs
 - **Stencils**: Up to **11.9 GFLOP/s**, **3.9×** faster, **83%** FPU util. (**≥15%** more than GPU code gens)
 - **SpMM**: Up to **6.4 GFLOP/s**, **4.6×** faster, **42%** FPU util. (**≥1.6×** more than sd. LA on GPUs)
 - **SpMSpM**: Up to **3.9 GCOMP/s**, **3.6×** faster, **49%** index comparator utilization



EPI FORUM



EuroHPC
Joint Undertaking

PLATINUM SPONSORS



EVIDEN



GOLD SPONSORS

