

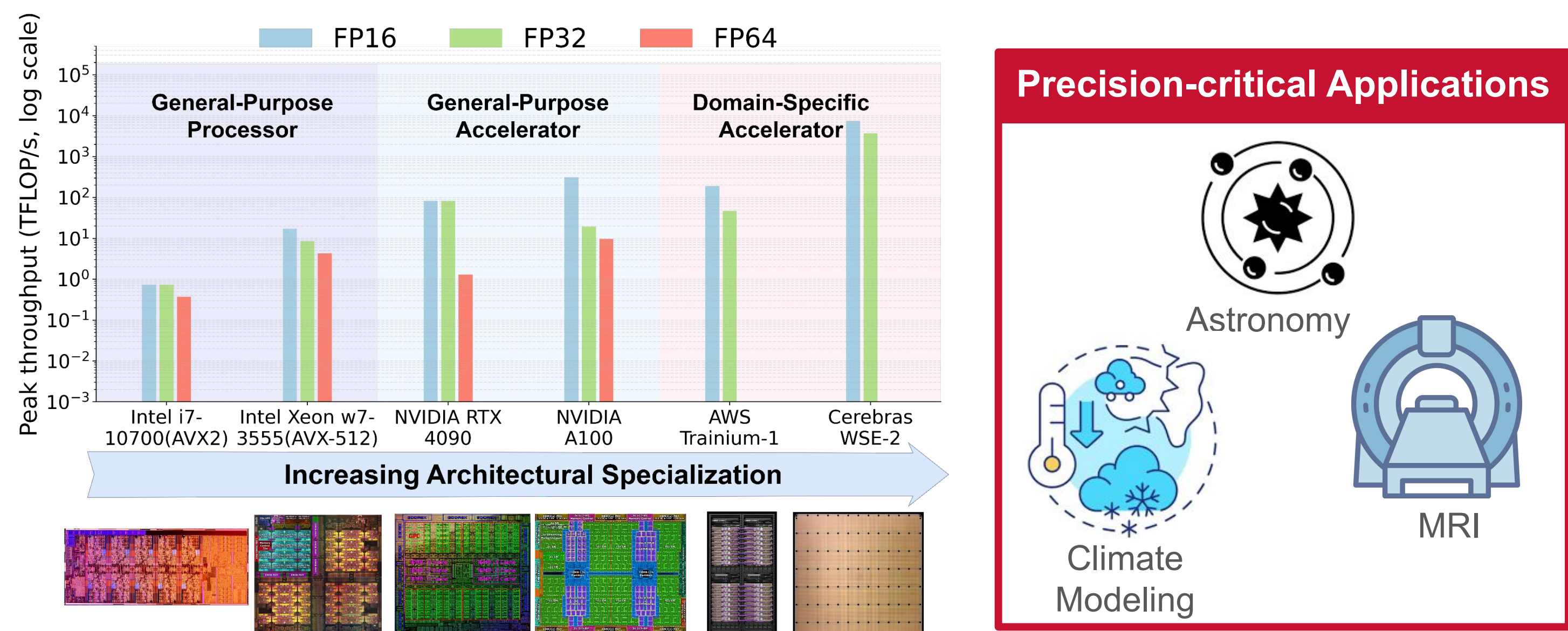
Double-Precision Floating-Point on AWS Trainium Using Integer Units: A First Look

Yunhao Lan, Larry Tang, Naifeng Zhang, Franz Franchetti; *Carnegie Mellon University*
IEEE High Performance Extreme Computing Conference (HPEC) 2026



Challenges

Trade-offs between Performance and Precision

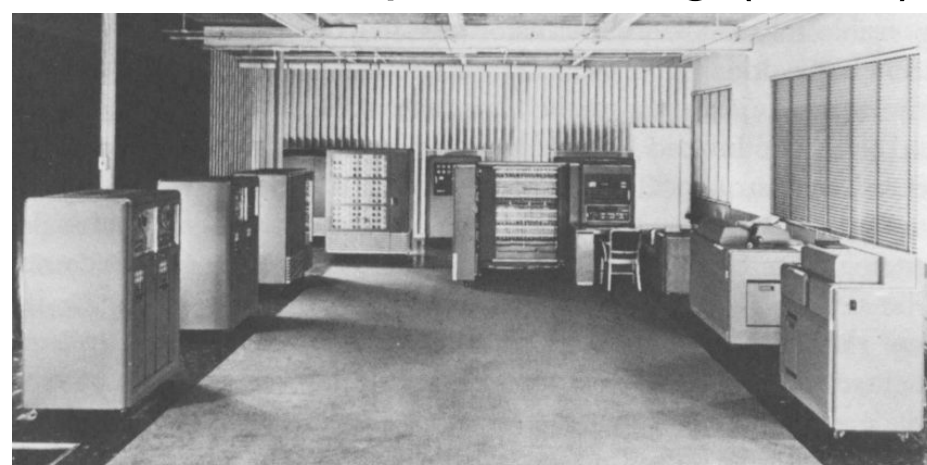


“CPUs + **limited-FP accelerators**” are increasingly unable to meet **scientific** HPC demands

Software FP Emulation: A Long History

Early systems

- EDSAC (1951)
- IBM 701 Speedcoding (1952)



IBM 701 defense calculator (1952)

Modern systems

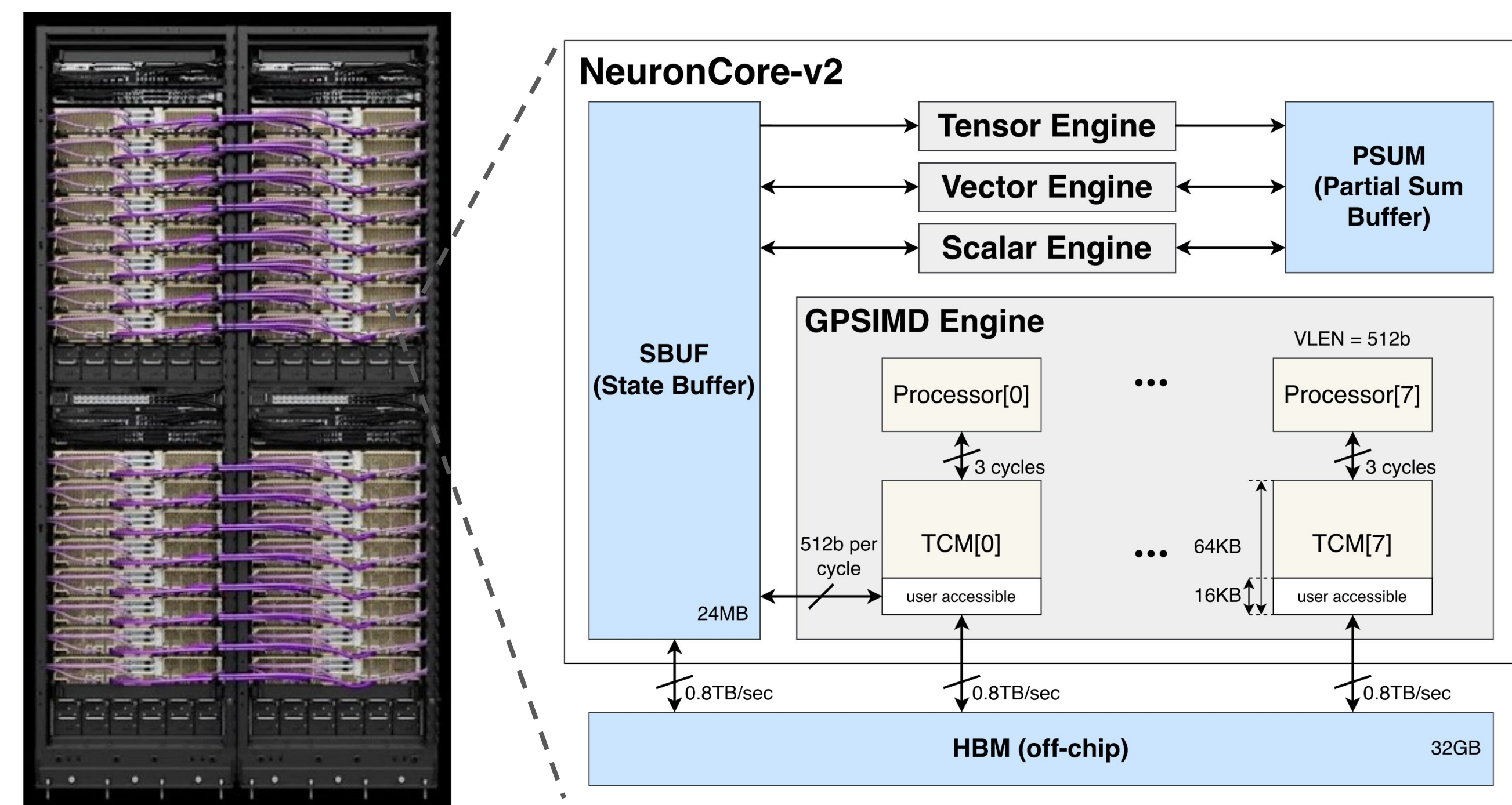
- GCC (libgcc) FP Emulation Routines
- Berkeley SoftFloat



ESP32-C3 embedded MCU (2020)

Software emulation can be used to provided **FP capabilities** not natively supported by hardware

AWS Trainium: No Native FP64

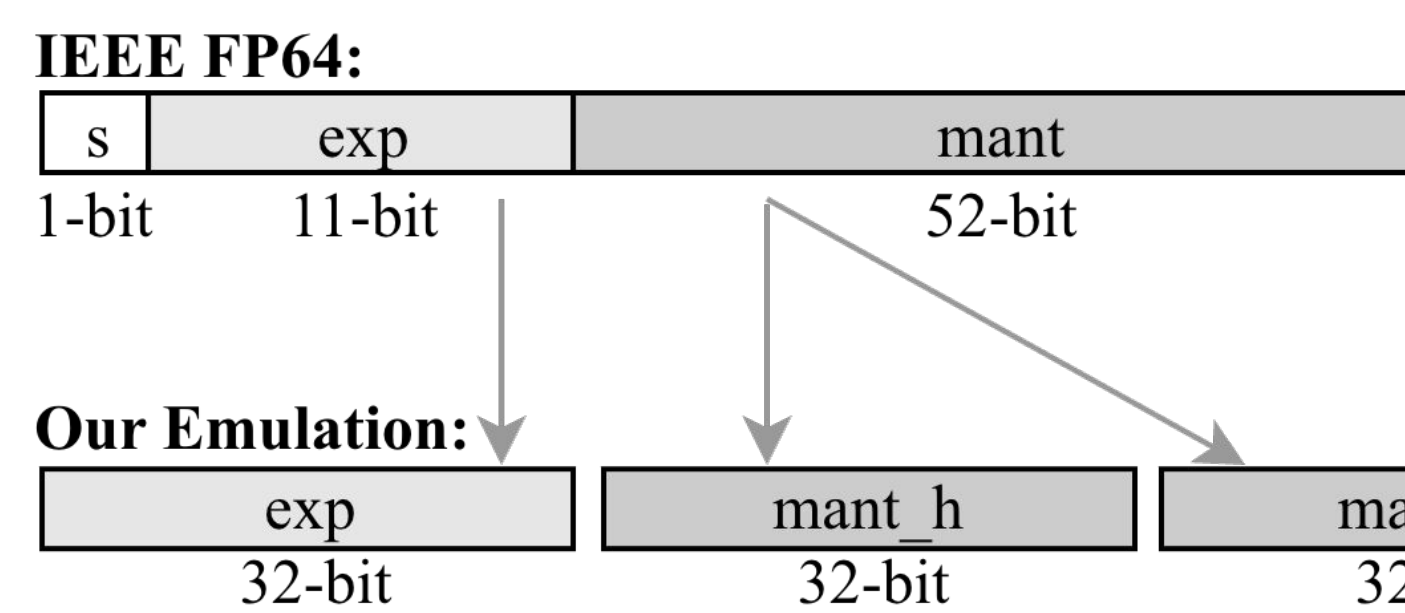


AWS Trainium 1:

- NeuronCore as the basic compute unit
- Four specialized compute engines per NeuronCore
- 32-bit native datapath; **no native FP64 support**
- Custom C++ operators provide low-level access to the GPSIMD engine

Method

Our Emulation on AWS Trainium



Supported operation includes:
emulated addition,
emulated subtraction,
emulated multiplication,
type conversion with FP64,
...

```
1 // _ce: output exponent in int32_t
2 // _cmh: output mantissa high in int32_t
3 // _cml: output mantissa low in int32_t
4 // _ae: input operand 1 exponent in int32_t
5 // _amh: input operand 1 mantissa high in int32_t
6 // _aml: input operand 1 mantissa low in int32_t
7 // _be: input operand 2 exponent in int32_t
8 // _bmh: input operand 2 mantissa high in int32_t
9 // _bml: input operand 2 mantissa low in int32_t
10 #define emulated_mul(_ce, _cmh, _cml, _ae, _amh,
    _aml, _be, _bmh, _bml) {
    // ...
    // Listing 2: Code snippet of the emulated floating-point addition
    // kernel on AWS Trainium targeting GPSIMD engine with XT-
    // IVP SIMD intrinsics.
```

Using Radix Kernels as FFT Building Blocks

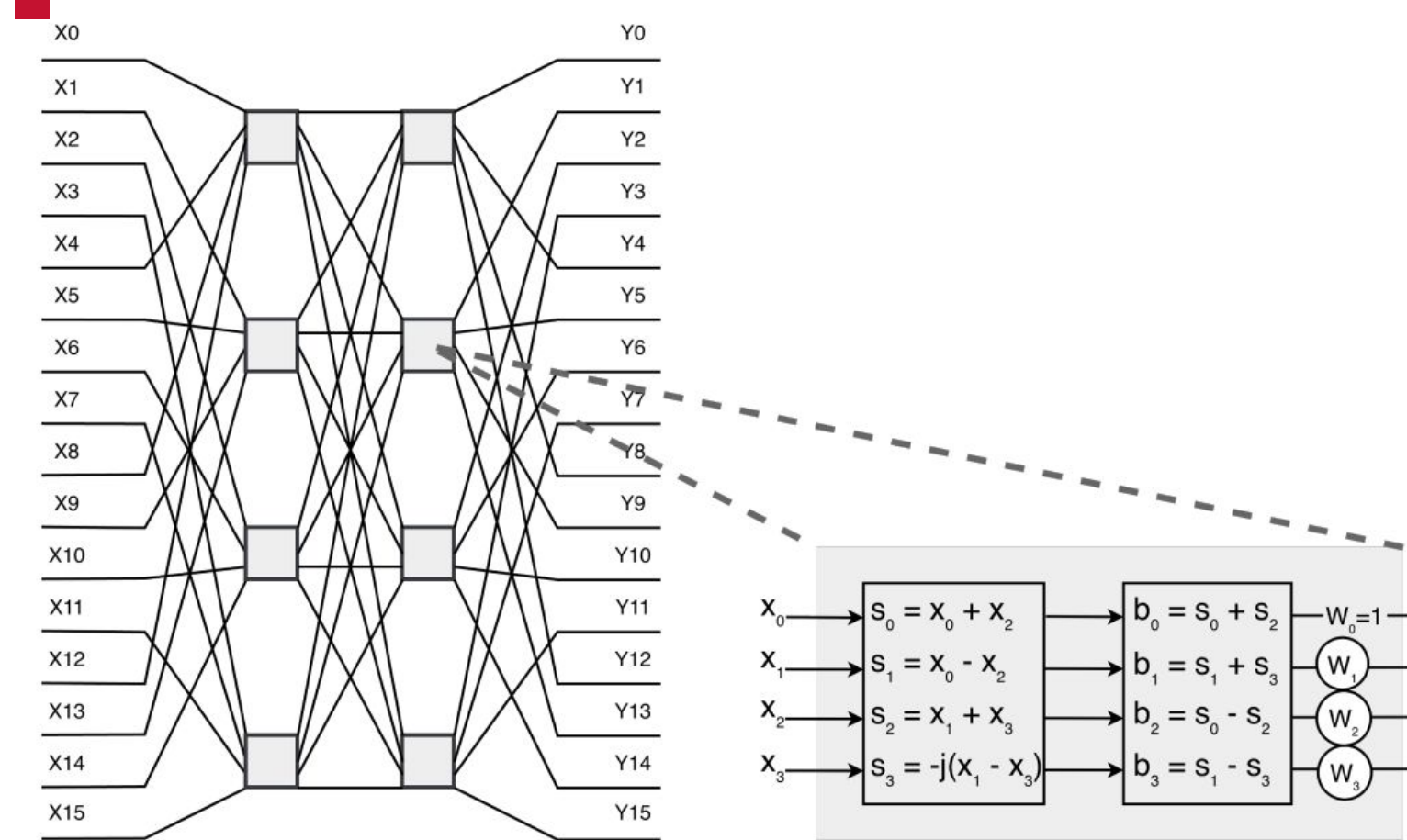


Fig. 2. A 16-point FFT composed of radix-4 kernels. Each stage applies multiple radix-4 kernels with stage-dependent twiddle factors. A radix-4 kernel operates on four complex values and serves as the reusable building block for larger FFTs, rather than as a standalone 4-point FFT.

Radix- r kernel: r -input butterfly + stage-dependent twiddles

Reusability: across stages and FFT sizes

FFT stage: multiple radix- r kernel instances

Full FFT: sequence of radix-kernel stages with data reordering

Emulated FP Radix Kernels on AWS Trainium

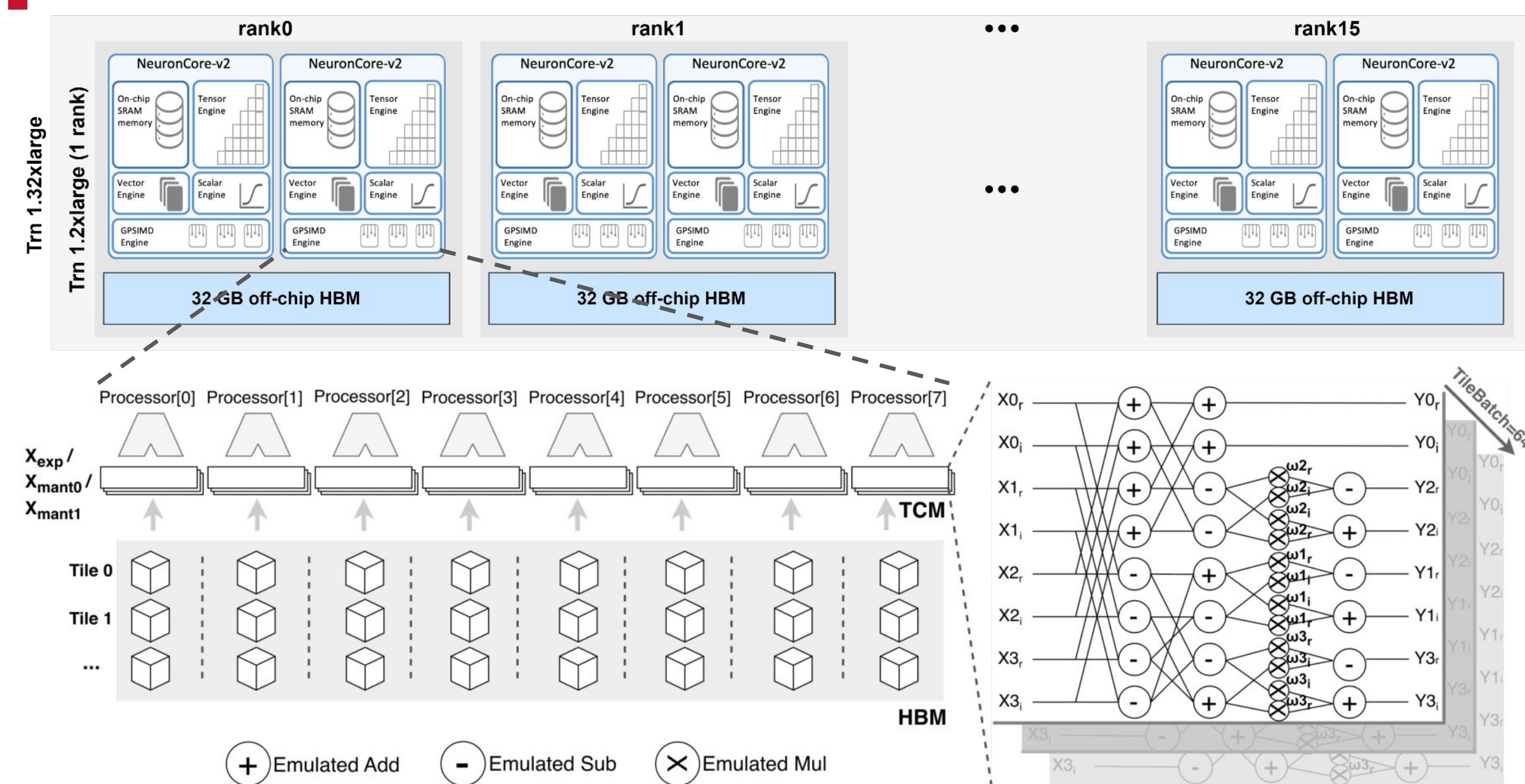


Fig. 3. Batched software-emulated floating-point radix-4 kernel implementation on gpsimdE using the custom C++ operator flow, where each gpsimdE processor loads one tile and precomputed twiddle factors from HBM into its private TCM, performs the radix-4 computation using our emulated operations, and stores the result back to HBM.

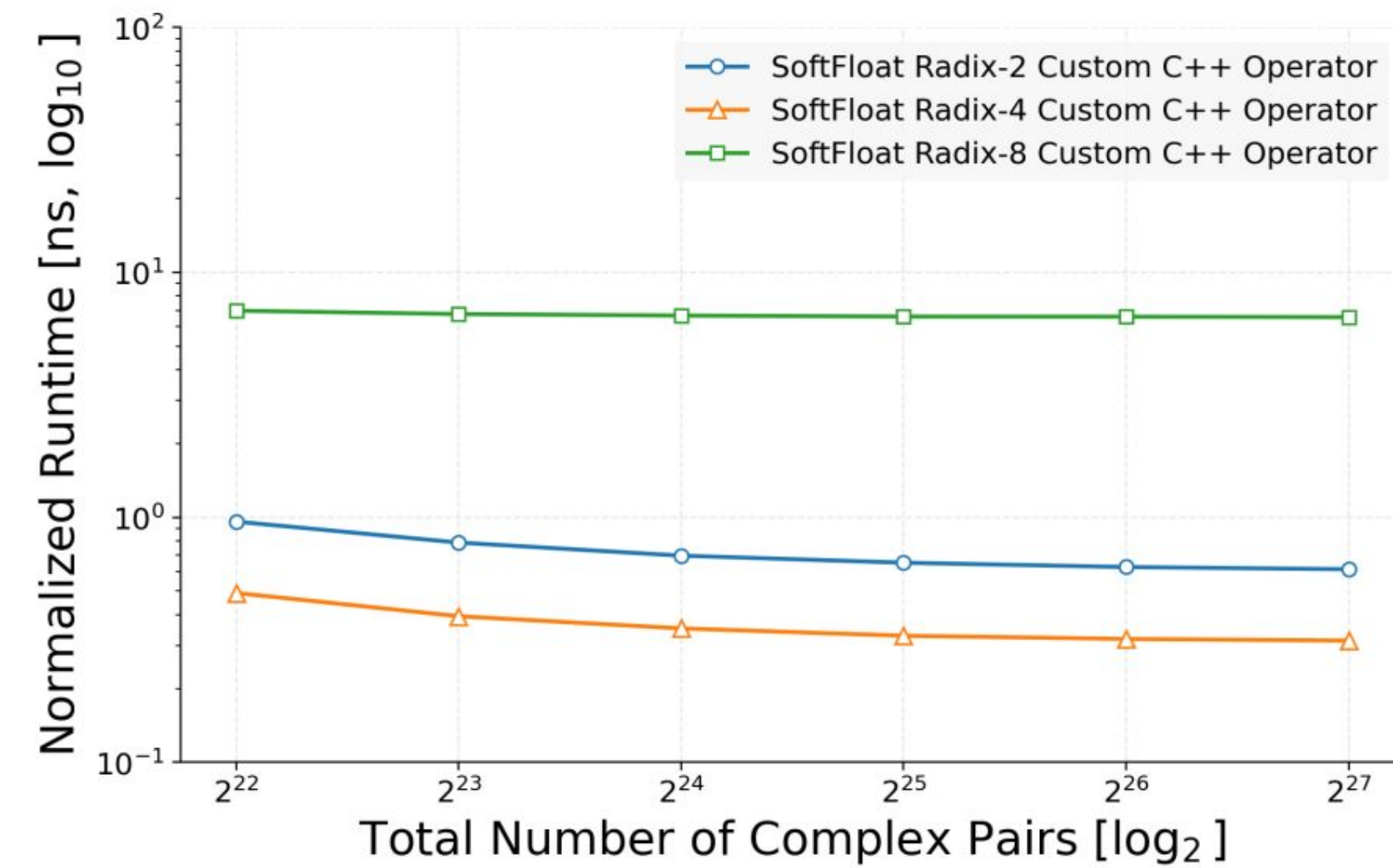
Batched parallelism ($32 \times 8 \times 16 = 4,096$ -way) across **3 hardware levels**:

- 32 NeuronCores**: 1 GPSIMD engine per NeuronCore
- 8 vector processors** per GPSIMD engine
- 16 × 32-bit lanes** per vector processor

Independent radix kernels mapped across all levels

Evaluation

Radix Size Evaluation



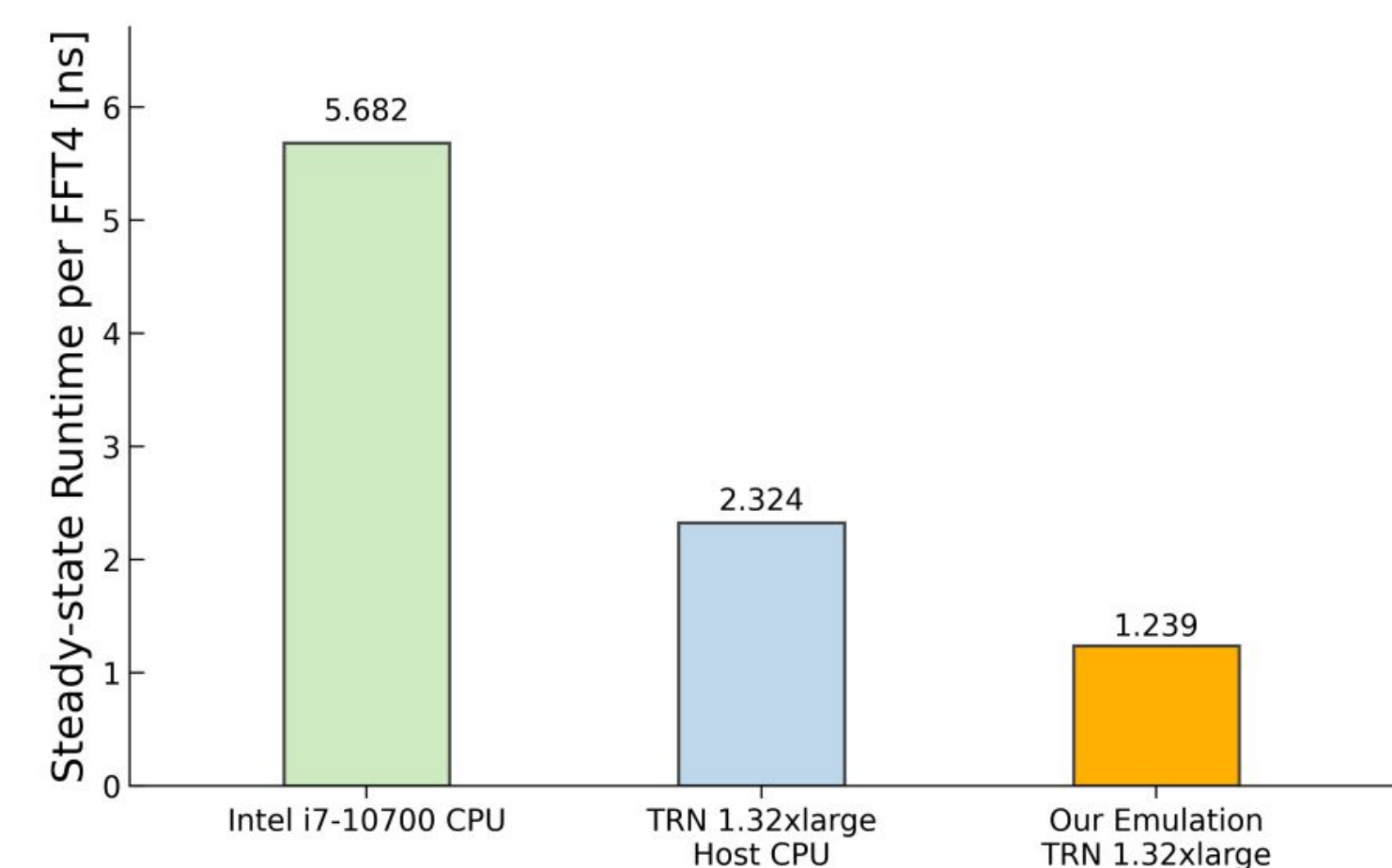
Trade-off between **arithmetic intensity** and **register pressure**.

- Larger radix gives:
- more **computation per load**
 - more **intermediates values**

Fig. 4. Normalized runtime of batched software-emulated floating-point radix-2, radix-4, and radix-8 kernels on gpsimdE. Runtime is normalized by the number of equivalent radix-2 butterfly operations to compare different radix sizes on the same basis.

With our emulation, radix-4 best balances **arithmetic intensity** and **register pressure**

Comparison with CPU FFT Baselines



Multi-threaded Intel MKL FFT on two platforms:

- Xeon Platinum CPU**: Trainium’s host CPU
- Core i7 CPU**: reference point for desktop-class CPU

Performance gain attributed to:

- high degree of **hierarchical parallelism**
- high-bandwidth** of its memory system

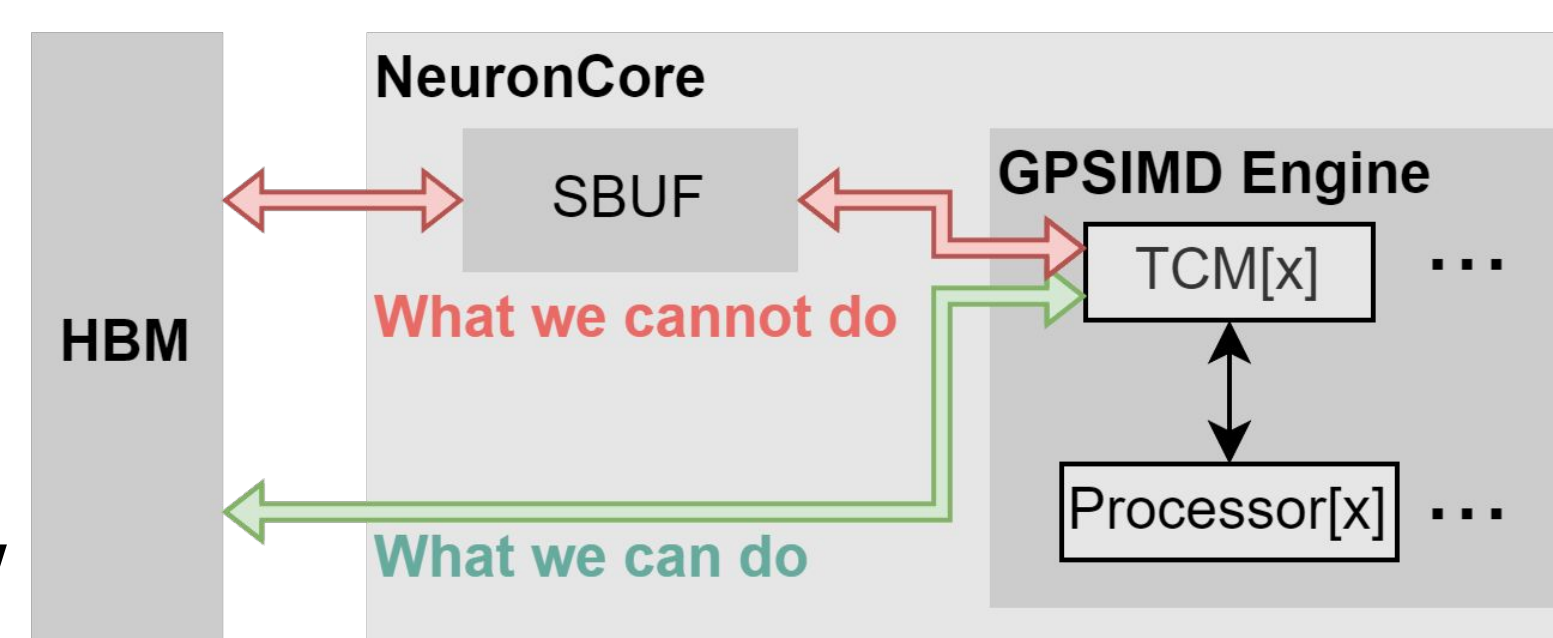
Fig. 5. Steady-state runtime per 1D 4-point FFT. CPU baselines use multi-threaded Intel MKL on the desktop Intel Core i7-10700 CPU and the trn1.32xlarge host Intel Xeon Platinum 8375C CPU, while the Trainium result uses our batched software-emulated floating-point radix-4 kernel on gpsimdE.

Offloading FFT butterflies to Trainium with our emulation benefits the overall “**CPU + accelerator**” system.

Scalability Limitation

For custom C++ operator flow:

- No SBUF access**: intermediate data cannot stay on-chip
- High data movement**: repeated HBM–TCM transfers b/w stages
- Compilation limit**: .text overflow beyond 1,024-point FFTs



Custom C++ operators suits **lightweight, independent** kernels

Future Work

- Move to the **Neuron Kernel Interface (NKI)** flow
- Exploit **SBUF-resident intermediate data** across stages enabled by NKI
- Build **full multi-stage FFT** pipelines