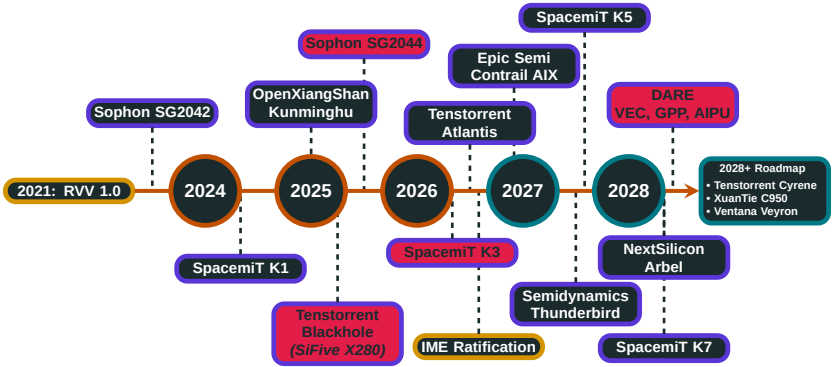
FAST FOURIER TRANSFORMS ON EMERGING RISC-V HARDWARE WITH THE RISC-V VECTOR EXTENSION

26th June 2026 | Daniel Seibel | Jülich Supercomputing Centre
Joint work with K. H. Mood, J. Badwaik, P. Chawla, S. Nassyr and A. Herten

Member of the Helmholtz Association

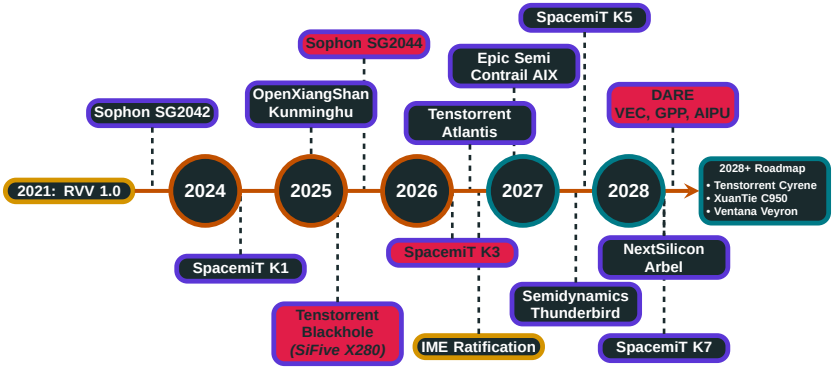
JÜLICH
Forschungszentrum

RISC-V in HPC is evolving



JÜLICH
Forschungszentrum

RISC-V in HPC is evolving



This talk presents our **juFFTe** library and shows

- how RVV offers new flexibility for HPC kernels
- how RVV 1.0-ready hardware is performing today

JÜLICH
Forschungszentrum

Digital Autonomy for RISC-V in Europe

DARE is funded by EuroHPC (€240M) and spans 38 partners

- Advancing EU's autonomy in HPC & AI with RISC-V
- 3 RISC-V-based chiplets (VEC, AIPU and GPP)
- Complete HPC & AI software stack for RISC-V
- Hardware/software co-design

The role of **Juelich Supercomputing Centre** is

- to co-lead software development (especially co-design)
- develops BLAS and FFT for RISC-V

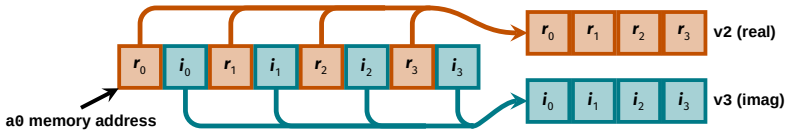
dare

Special features of RISC-V Vector Extension for FFT

RVV's special features offer new ways of coding FFT kernels

- Vector-length-agnostic code `vlsetvli vl,AVL,…`

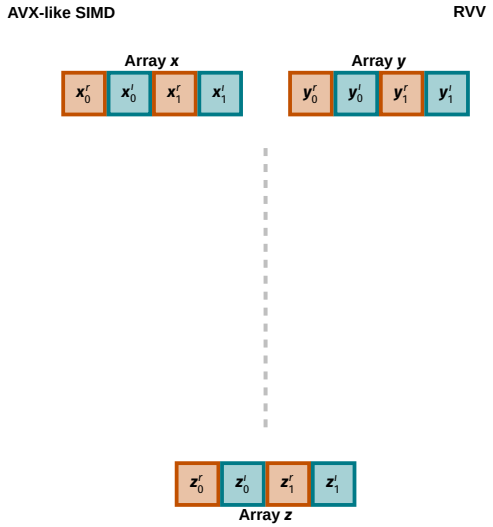
- Segmented loads `vlseg2 v2,(a0)` (also stores)



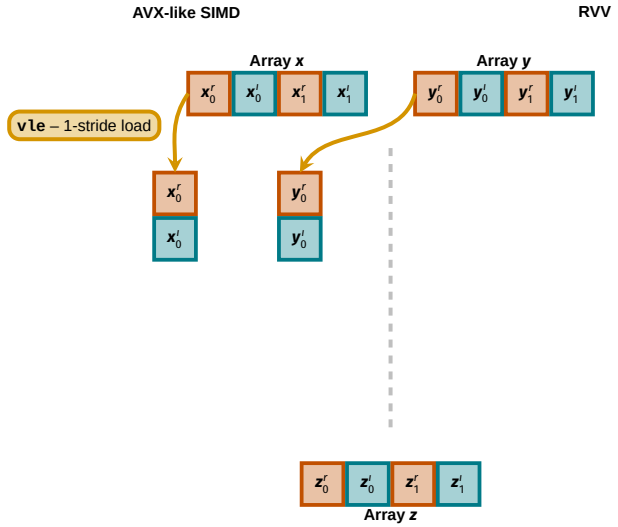
- Permutation instructions (`vrgather`, `vslide`)

JÜLICH Forschungszentrum

Multiplying complex numbers with RVV

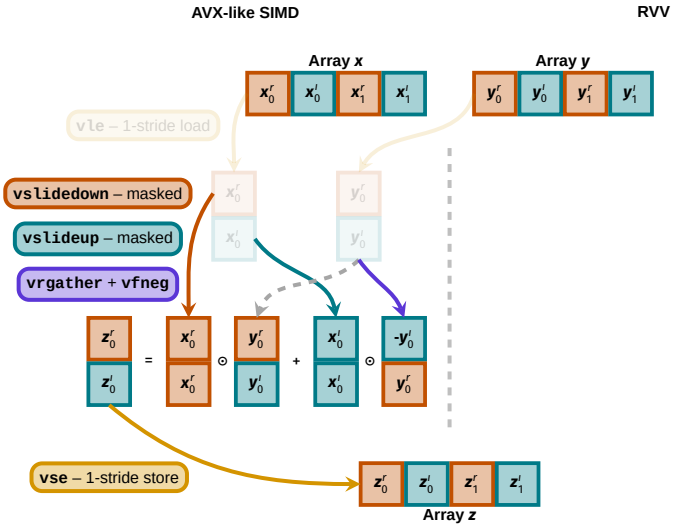


Multiplying complex numbers with RVV



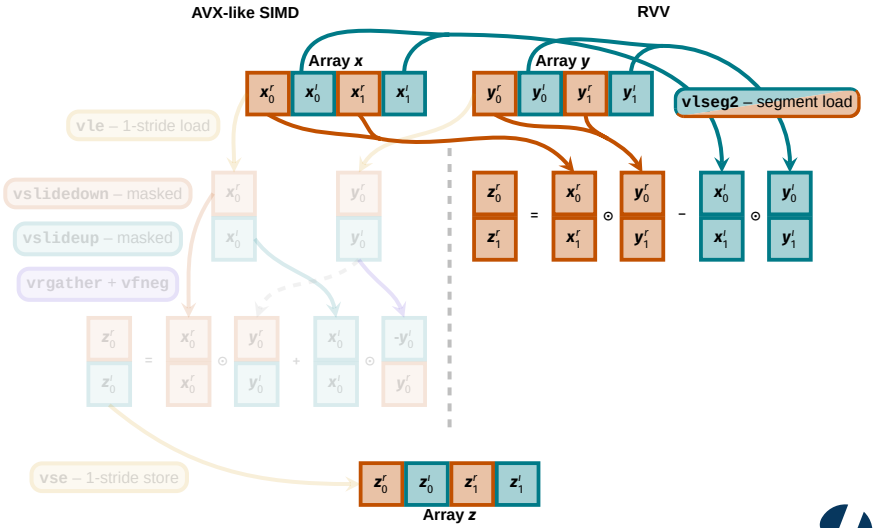
JÜLICH Forschungszentrum

Multiplying complex numbers with RVV



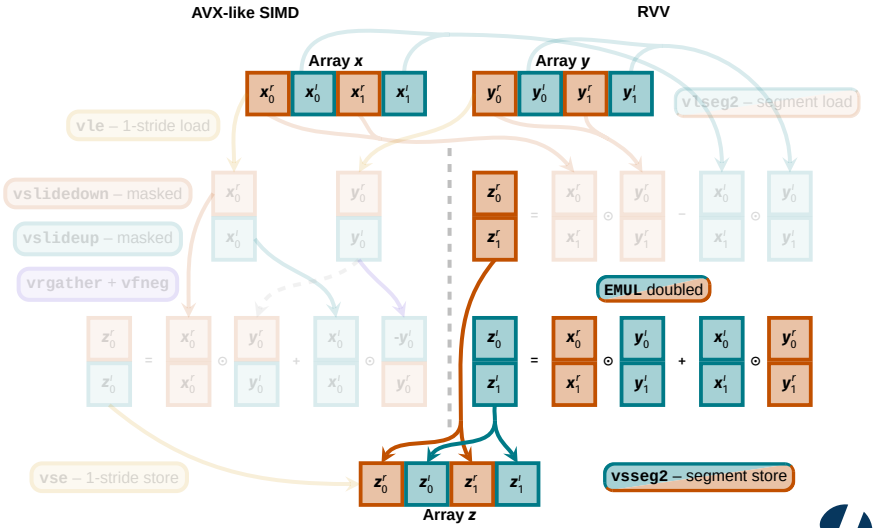
JÜLICH Forschungszentrum

Multiplying complex numbers with RVV



JÜLICH Forschungszentrum

Multiplying complex numbers with RVV



JÜLICH Forschungszentrum

Benchmark setup

Our new library **juFFTe** (`https://github.com/FZJ-JSC/juFFTe`):

- High-performance yet lightweight
- Optimized for RVV 1.0
- Easy access to low-level kernels allows easy experimentation
- FFTW3 interface

We compare the performance against FFTW3[1]:

- 1D c2c FFT chosen as benchmark
- juFFTe uses segmented, FFTW3 traditional vectorization

[1] `https://github.com/rdolbeau/fftw3/tree/riscv-v-clean`

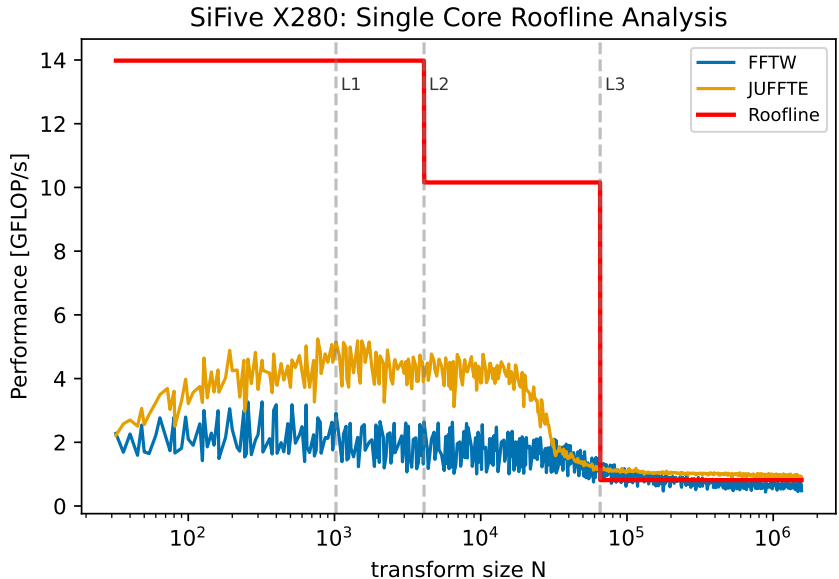
SiFive X280 @ JSC's Tenstorrent Blackholes

Core Specifications:

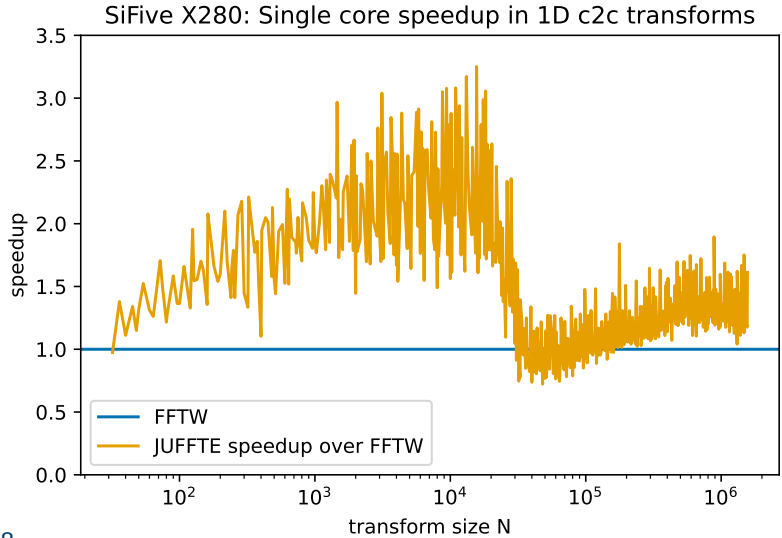
- **Arch:** 8-stage dual-issue in-order
- **Cores:** 4 Cores @ 1.75 GHz
- **Vector:** VLEN 512 bit (DLEN 256 bit)
- **Peak FP64:** 13.98 GFLOP/s per core
- **L1 Cache:** 32 kiB per core @ 55.90 GB/s
- **L2 Cache:** 128 kiB per core @ 55.82 GB/s
- **L3 Cache:** 2 MiB per 4 cores @ 25.00 GB/s

Tenstorrent Galaxy (Photo by Sebastian Achilles)

SiFive X280 – Single Core



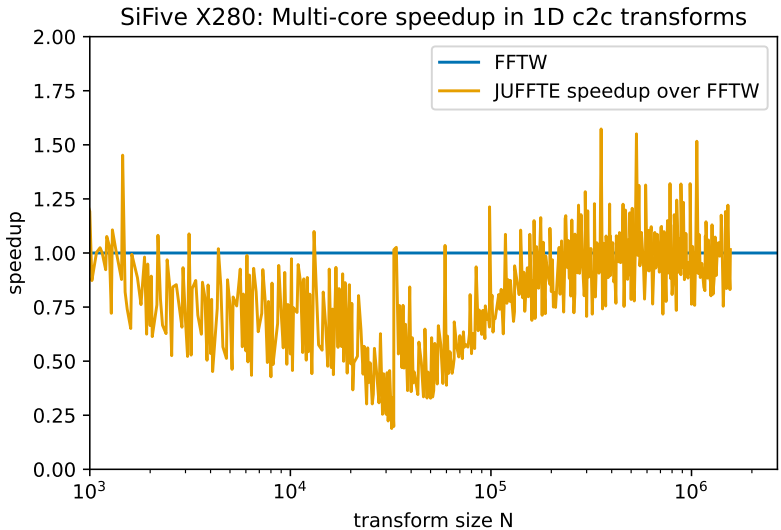
SiFive X280 – Single Core



- Segmented loads and stores are very fast on X280
- juFFTe's cache-blocking does not amortize at L3 boundary

JÜLICH Forschungszentrum

SiFive X280 – Multi Core



- FFTW multi-threaded algorithms handle small sizes better
- For large sizes performance is similar

JÜLICH Forschungszentrum

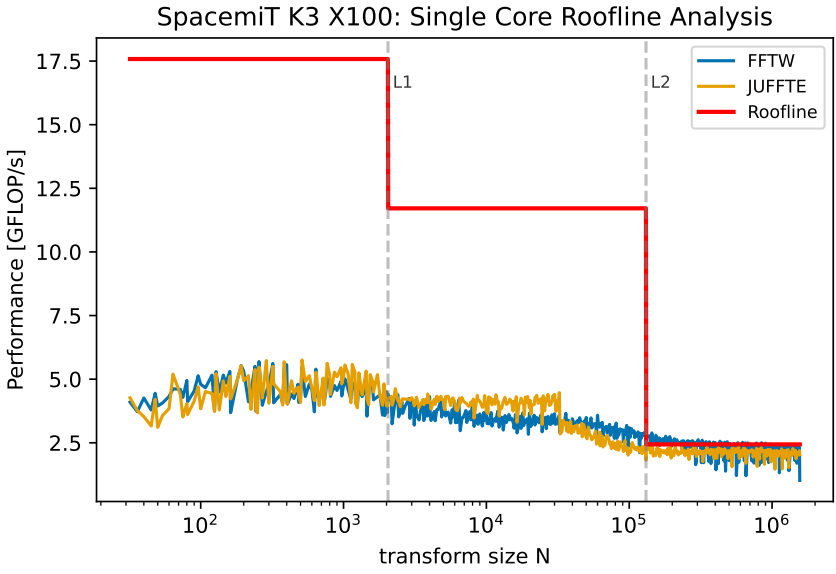
SpacemiT K3 (X100) @ SpacemiT Cloud

Core Specifications:

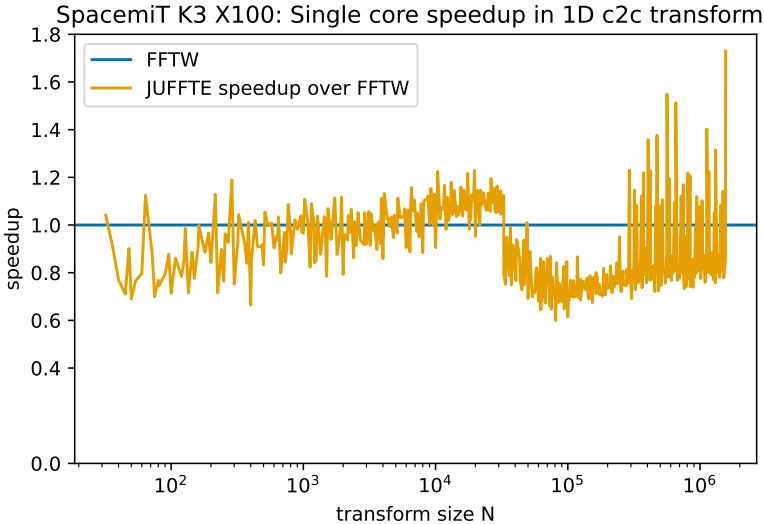
- **Arch:** 4-issue out-of-order
- **Cores:** 8 Cores @ 2.4 GHz
- **Vector:** VLEN 256 bit (DLEN 128 bit)
- **Peak FP64:** 17.58 GFLOP/s per core
- **L1 Cache:** 64 kiB per core @ 44.51 GB/s
- **L2 Cache:** 4 MiB per 4 cores @ 28.83 GB/s
- **L3 Cache:** n/a

SpacemiT K3 (Photo by Stepan Nassyr)

SpacemiT X100 – Single Core



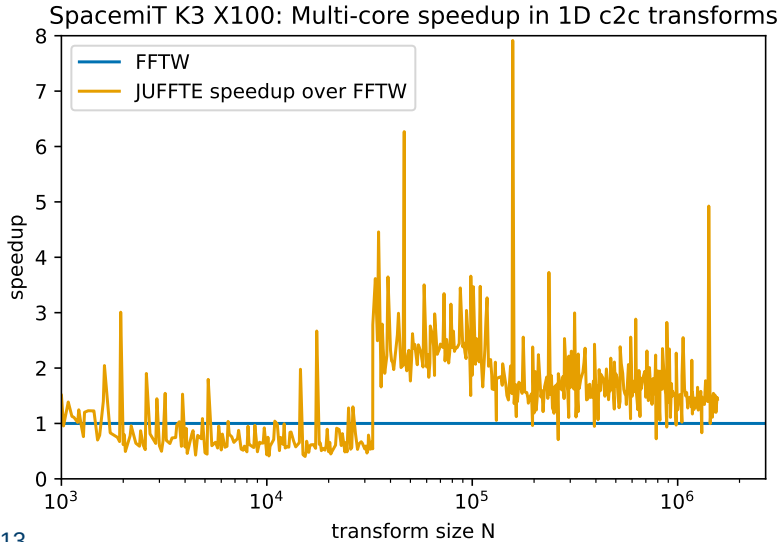
SpacemiT X100 – Single Core



- Segmented vectorization leads in L2
- juFFTe's cache-blocking does not always amortize outside L2

JÜLICH Forschungszentrum

SpacemiT X100 – Multi Core



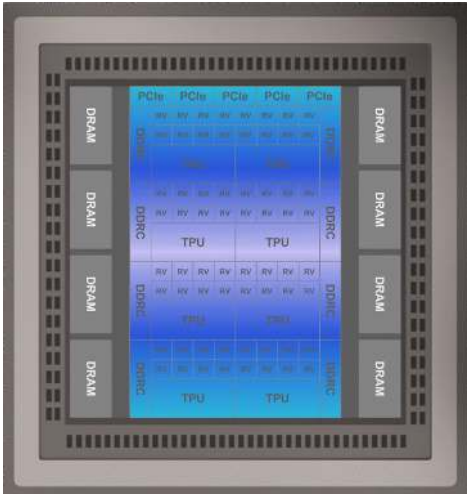
- FFTW auto-tuning set to `ESTIMATE`
- With multiple cores, cache-blocking proves beneficial

JÜLICH Forschungszentrum

Sophon SG2044 (Xuantie C920v2) @ Monte Cimone v3

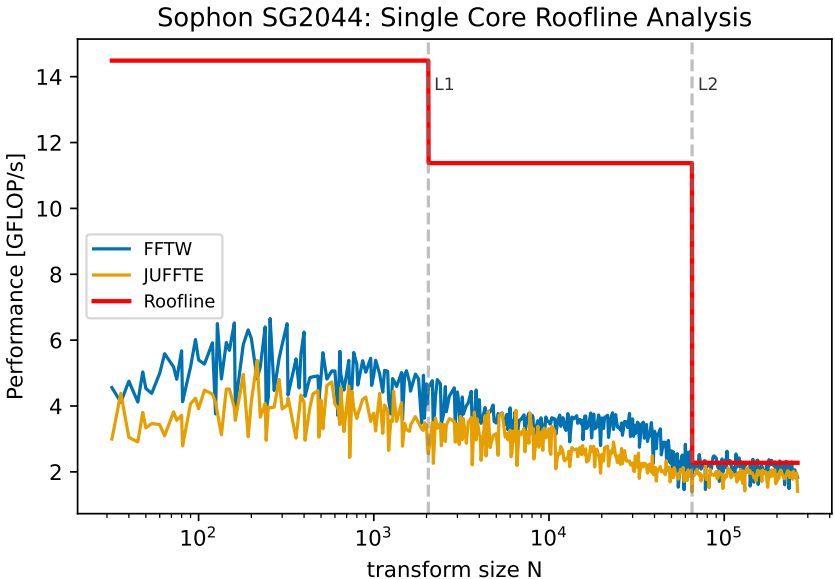
Core Specifications:

- **Arch:** 12-stage multiple-issue out-of-order
- **Cores:** 64 Cores @ 2.6 GHz
- **Vector:** VLEN 128 bit (DLEN 128 bit)
- **Peak FP64:** 20.07 GFLOP/s per core
- **L1 Cache:** 64 kiB per core @ 35.66 GB/s
- **L2 Cache:** 2 MiB per 2 cores @ 28 GB/s
- **L3 Cache:** 64 MiB per 64 cores @ 5.6 GB/s

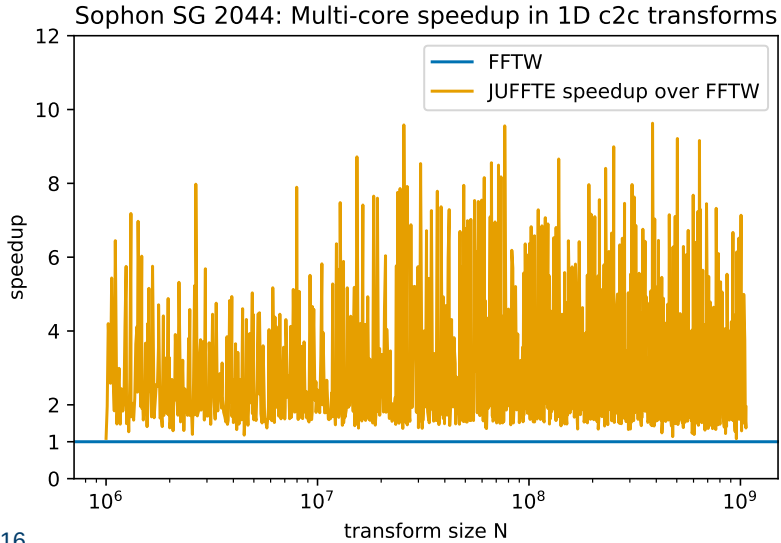


Sophgo SG2044 block diagram (Photo by Sophgo)

Sophon SG2044 – Single Core



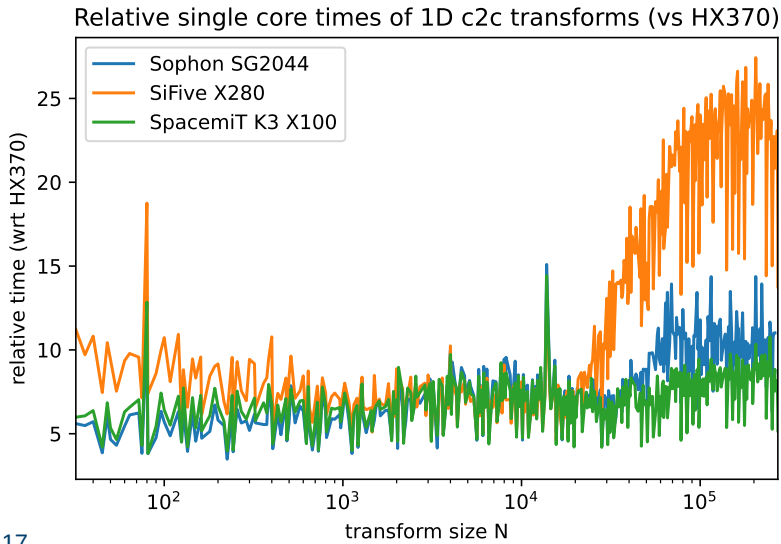
Sophon SG2044 – Multi Core (Large transforms)



- FFTW auto-tuning set to `ESTIMATE`
- For large sizes, juFFTe outperforms FFTW

JÜLICH Forschungszentrum

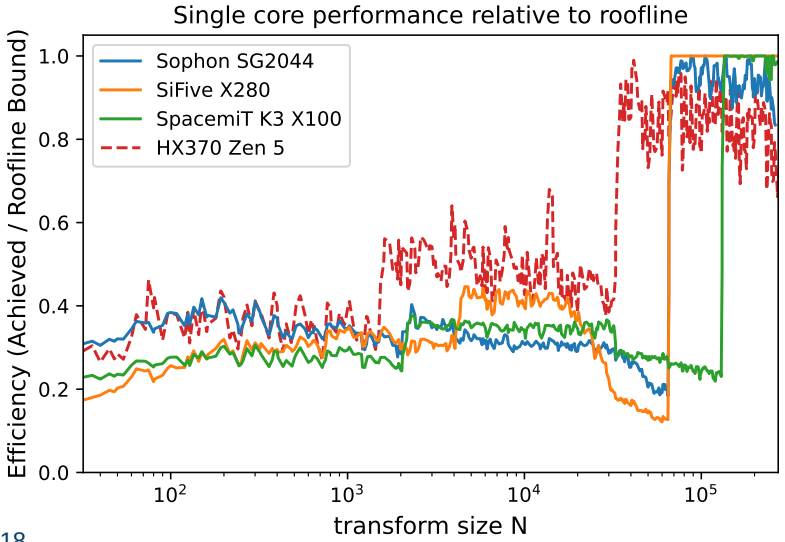
Comparison against AMD HX370 (Zen 5, AVX-512)



- The HX370 is capped at @ 2.6 GHz
- RISC-V is able to achieve up to 30% of HX370 performance

JÜLICH Forschungszentrum

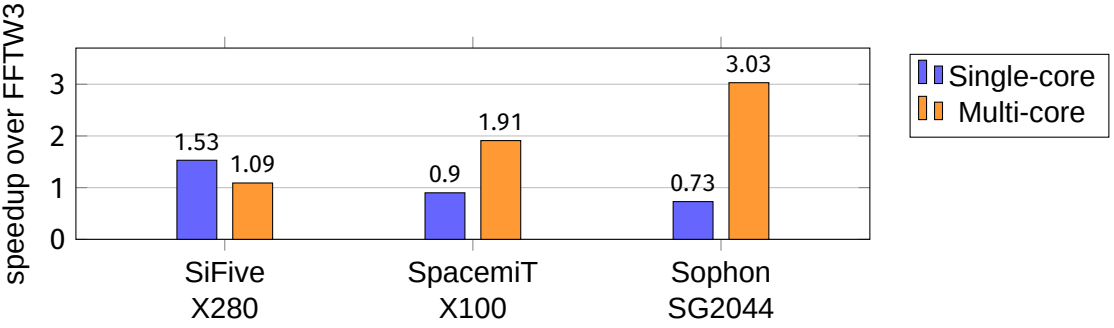
Comparison against AMD HX370 (Zen 5, AVX-512)



- Relative to roofline, the performance is comparable

JÜLICH Forschungszentrum

Conclusion



Takeaway:

- Multiple RVV strategies needed to cover all hardware ($\Longrightarrow$ **radixgen**)
- For FFTs, hardware needs fast segmented loads/stores or gather/slides

juFFTe on GitHub: `https://github.com/FZJ-JSC/juFFTe`

Thank you!

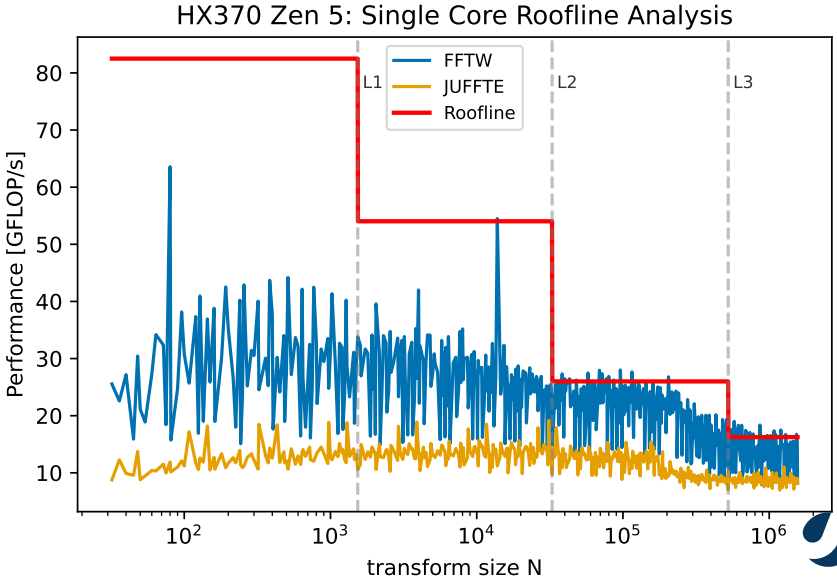
Email: d.seibel@fz-juelich.de

The authors gratefully acknowledge

- the Monte Cimone project for access to the Sophon SG2044
- SpacemiT and the RISC-V Ecological Application Innovation Center in Zhujiang for remote early access to the SpacemiT K3
- Funding for parts of this work has been received from EuroHPC's project DARE SGA 1 under Grant Agreement No. 101202459

JÜLICH Forschungszentrum

HX370 Zen 5 – Single Core



JÜLICH Forschungszentrum

Fast Fourier Transforms

Let $N = 2m$. A radix-2 FFT kernel performs:

1. Butterfly multiplication:

$$Y = \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} \begin{pmatrix} x_0 & \cdots & x_{m-1} \\ x_m & \cdots & x_{2m-1} \end{pmatrix}$$

2. Twiddle factor multiplication:

$$Z = \begin{pmatrix} 1 & 1 & \cdots & 1 \\ 1 & \omega_N & \cdots & \omega_N^{m-1} \end{pmatrix} \odot Y, \quad \omega_N = \exp(-2\pi\imath/N)$$

3. Shuffle / transposition of Z for next kernel