GREAT EXPECTATIONS

Benchmarking the Real-World Performance of RVV 1.0 in HPC

June 26, 2026 | Prateek Chawla (for colleagues) | Jülich Supercomputing Center

Introduction

RVV FOR HPC?

RVV 1.0 presents some interesting features:

- Vector-Length Agnostic programming model: Allows both portability and scalability
- Flexible register configurations, gather/scatter and permutation operations
  - Useful for complex data movement; kernels using dense linear algebra, stencil compute, memory-bound workloads.
- Emerging RISC-V hardware accelerators and HPC-focused implementations
  - Growing ecosystem of vector-capable processors (SiFive, SpacemiT, SOPHGO, etc.)
  - Momentum towards production-grade HPC systems with RVV support
- DARE project: Developing a complete RISC-V HPC stack
  - EU-funded initiative advancing RISC-V ecosystem for HPC
  - Supporting hardware development; compiler, runtime, and application optimization

RVV FOR HPC?

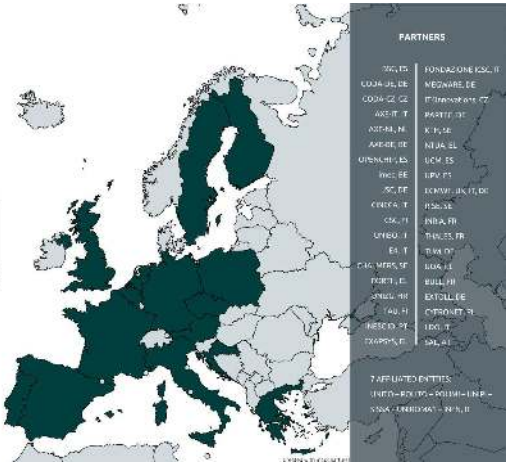
RVV 1.0 presents some interesting features:

- Vector-Length Agnostic programming model: Allows both portability and scalability
- Flexible register configurations, gather/scatter and permutation operations
  - Useful for complex data movement; kernels using dense linear algebra, stencil compute, memory-bound workloads.
- Emerging RISC-V hardware accelerators and HPC-focused implementations
  - Growing ecosystem of vector-capable processors (SiFive, SpacemiT, SOPHGO, etc.)
  - Momentum towards production-grade HPC systems with RVV support
- DARE project: Developing a complete RISC-V HPC stack
  - EU-funded initiative advancing RISC-V ecosystem for HPC
  - Supporting hardware development; compiler, runtime, and application optimization

Question

How ready is RISC-V for HPC?

THE DARE PROJECT



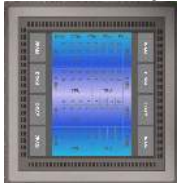
Devices and benchmarks

DEVICES

SpacemiT K1
Image courtesy: spacemit.com

SpacemiT K3
Image courtesy: Stepan Nassyr

NVIDIA Grace
Image courtesy: nvidia.com



SOPHGO SG2044
Image courtesy: SOPHGO

SiFive X280
Image courtesy: sifive.com

DEVICE DETAILS

		SpacemiT			Sophon	TT	NVIDIA
Metric	Unit	**K1**	**K3**	**K3**	**SG2044**	**Blackhole**	**Grace**
Arch.		X60	X100	A100	C920v2	X280	Neoverse V2
Release Date		04/2024	04/2026	04/2026	02/2025	04/2025	05/2023
#Cores		8	8	8	64	4x4	72
VLEN	bit	256	256	1024	128	512	128
DLEN	bit	256	256	256 (FP64) 512 (FP32)	256	256	512
Peak Perf (FP64)	GFLOP/s	102.4	140.8	155.2	1331.2	224	3550
Peak Perf/core (FP64)	GFLOP/s	12.8	17.6	14.4	20.8	14	55.3
Peak Perf/core (FP32)	GFLOP/s	25.6	35.2	57.6	41.6	28	110.6
L1d Cache	KiB	32	64	64	64	32	64
L2 Cache	KiB	512/4c	8192/8c	8192/8c	2048/4c	128/1c	1024/1c
L3 Cache	MiB	-	-	-	64/64c	2/4c	114/72c
Clock Freq.	GHz	1.6	2.2	1.8	2.6	1.75	3.1 (max. 3.456)
Frontend		2-way iO	4-way OoO	2-way iO	3-dispatch/ 4-commit/ 8-issue OoO	2-way iO	6-wide OoO

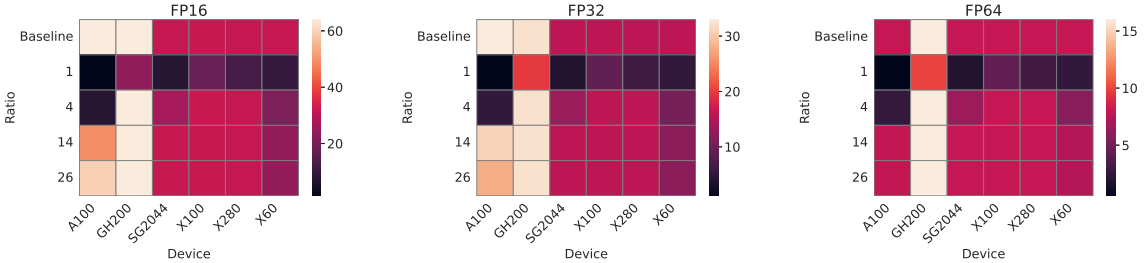
SELECTED BENCHMARK WORKLOADS

Curated workloads to comprehensively evaluate system performance across various operational parameters and stress conditions.

FMA Throughput	Investigate potential port contention between VLSU and VFPU
STREAM	Memory bandwidth
BLAS	DGEMM and SGEMM performance and compute efficiency
FFTW	1D c2c FFT, and `libbench2` performance
HPL	Compute-bound dense linear algebra
HPCG	Sparse iterative solver, stresses memory bandwidth and network latency

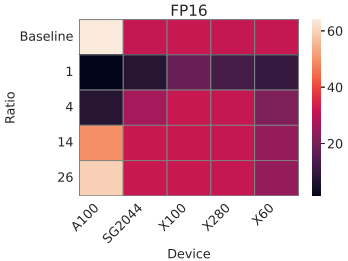
Results

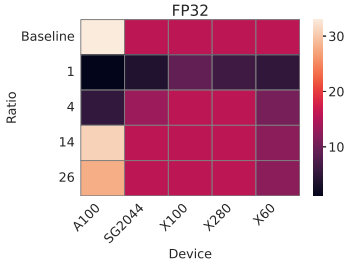
FMA CHAIN THROUGHPUT: COMPARISON WITH NVIDIA GRACE

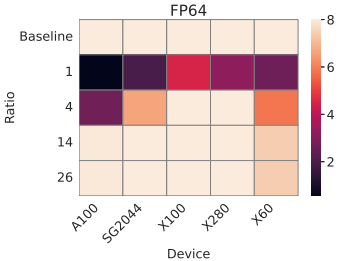


- Stress memory latency
- Execute `vfmacc` and `vle` at a specific ratio
- Grace: performance gap remains

FMA CHAIN THROUGHPUT: RISC-V DEVICES





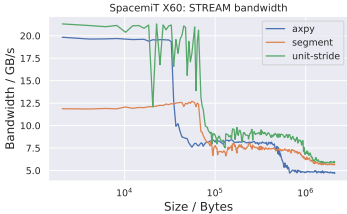


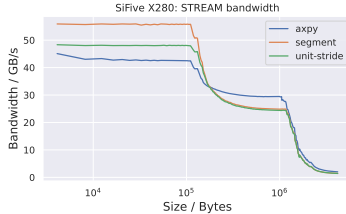
- RISC-V devices show memory latency issues in FP64 workloads

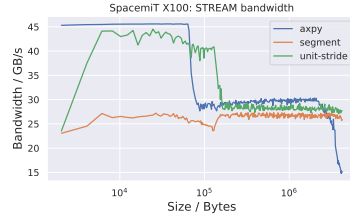
FMA CHAIN THROUGHPUT: RAW DATA

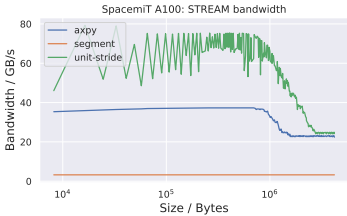
		SpacemiT			Sophon	TT	NVIDIA
vfmacc vle	**Data Type**	**K1** X60	**K3** X100	**K3** A100	**SG2044** C920v2	**Blackhole** x280	**Grace** Neoverse V2
Baseline	**FP64**	7.999	7.999	7.999	7.9965	7.9871	15.9999
1	**FP64**	2.6624	4.517	0.5593	1.9989	3.2171	9.9426
4	**FP64**	5.8561	8.0	2.6987	6.6409	7.9873	15.9999
14	**FP64**	7.4184	8.0	7.9238	7.9968	7.9875	15.9999
26	**FP64**	7.4176	8.0	7.9238	7.9973	7.9873	15.9999
Baseline	**FP32**	15.997	15.999	32.985	15.9926	15.9742	31.9998
1	**FP32**	5.3256	9.143	1.1441	3.9979	6.3894	19.8889
4	**FP32**	10.8080	15.999	5.482	13.4759	15.9746	31.9998
14	**FP32**	12.3996	15.999	30.7966	15.9918	15.9743	31.9998
26	**FP32**	12.3986	15.999	27.938	15.9924	15.9742	31.9998
Baseline	**FP16**	31.9491	31.998	63.9966	31.9778	31.9493	63.9997
1	**FP16**	10.6514	18.286	1.4329	7.9962	12.779	23.9999
4	**FP16**	21.6148	31.998	7.9228	27.1128	31.9479	63.9997
14	**FP16**	24.7973	31.998	49.4824	31.9859	31.9482	63.9997
26	**FP16**	24.7967	31.998	59.4777	31.9882	31.948	63.9997

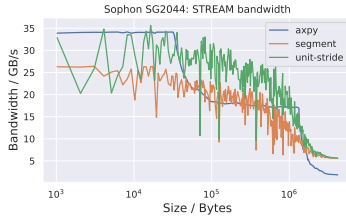
STREAM

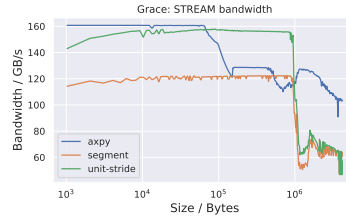

SpacemiT X60: STREAM bandwidth
Bandwidth / GB/s
Size / Bytes
axpy
segment
unit-stride


SiFive X280: STREAM bandwidth
Bandwidth / GB/s
Size / Bytes
axpy
segment
unit-stride

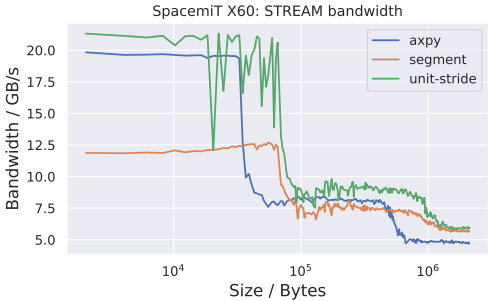

SpacemiT X100: STREAM bandwidth
Bandwidth / GB/s
Size / Bytes
axpy
segment
unit-stride

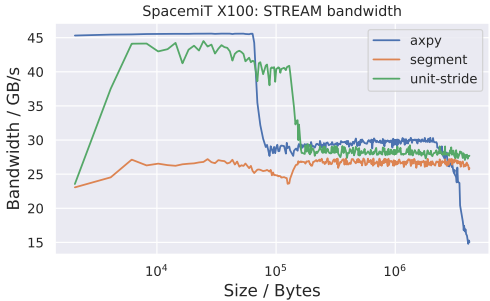

SpacemiT A100: STREAM bandwidth
Bandwidth / GB/s
Size / Bytes
axpy
segment
unit-stride


Sophon SG2044: STREAM bandwidth
Bandwidth / GB/s
Size / Bytes
axpy
segment
unit-stride


Grace: STREAM bandwidth
Bandwidth / GB/s
Size / Bytes
axpy
segment
unit-stride

STREAM





- Inter-generational data shows a definitive improvement in memory bandwidth
- Segmented load/store kernel stability highlights improvements made to data bus in the X100

A RATHER QUICK INTRODUCTION TO BLIS

- BLAS-Like Instantiation Software framework
- Isolates kernels that can be optimized
- Provides a fully functional reference BLAS
- Allows advanced multithreading

https://github.com/flame/blis

BLIS: EXAMPLE GEMM PSEUDOCODE

```
for(jc =0; jc < n; jc += nc){ // Loop L1
    for(pc =0; pc < k; pc += kc){ // L2
    Bc = pack_B(pc:pc+kc-1, jc:jc+nc-1); // packing kernel for B: fits into L3 cache
    for(ic =0; ic < m; ic += mc){ // L3
        Ac = pack_A(ic:ic+mc-1, pc:pc+kc-1); // packing kernel for A: fits into L2 cache
        for(jr =0; jr < nc; jr += nr){ // L4
            for(ir =0; ir < mc; ir += mr){ // L5
               // L6: Microkernel, 0:kc-1 chunk fits into L1 cache
               // Microkernel must fit entirely in registers
                C[ic+ir:ic+ir+mr-1, jc+jr:jc+jr+nr-1] +=
                    Ac[ic+ir:ic+ir+mr-1, 0:kc-1] *
                    Bc[0:kc-1, jc+jr:jc+jr+nr-1]
                }
            }
        }
    }
}
```

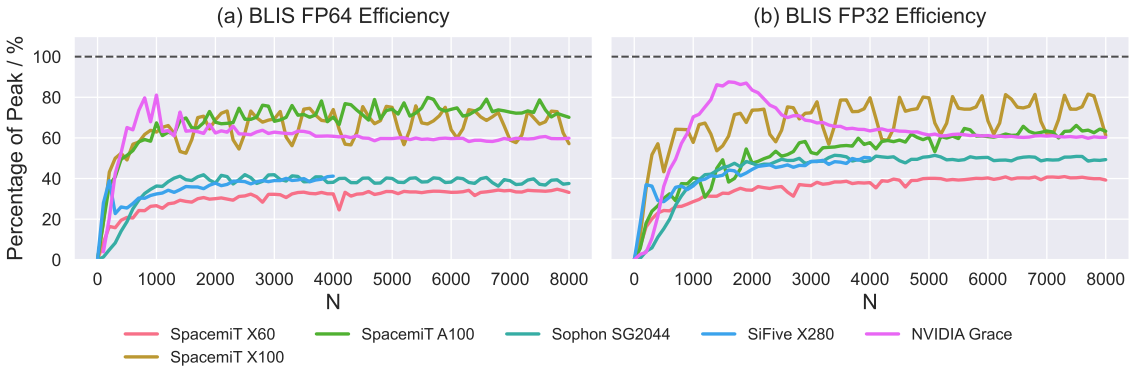
BLAS BENCHMARKS: SETUP

BLIS Threading Config

Platform	**#Threads**	J_C	I_C	J_R	I_R
X60	8	2	1	4	1
X100	8	2	1	4	1
A100	8	1	1	8	1
X280	4	1	1	4	1
SG2044	64	1	4	16	1
Grace	72	NVPL (Auto)			

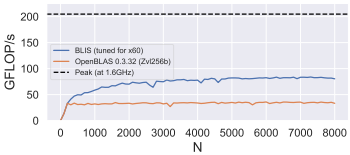
I_R, J_R: Loops around the microkernel
I_C, J_C: Loops around packing kernels

BLAS BENCHMARKS: EFFICIENCY

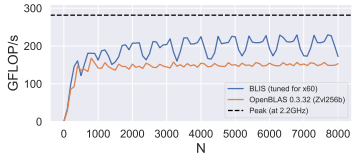


- SpacemiT X100 and A100 cores sustain a higher peak performance at large FP64 and FP32 DGEMM workloads compared to NVIDIA Grace.

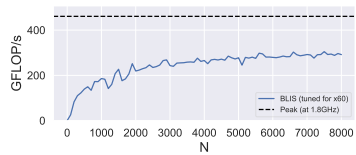
BLAS BENCHMARKS: SGEMM



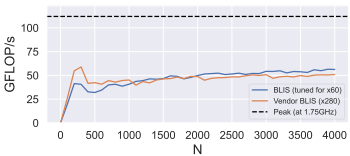
X60



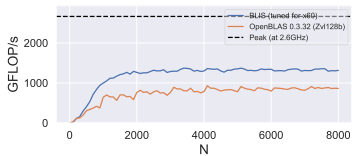
X100



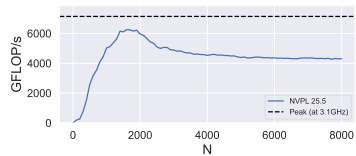
A100



X280

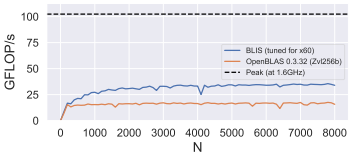


SG2044

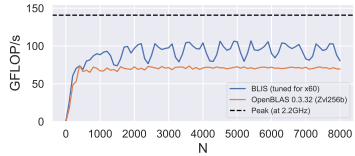


Grace

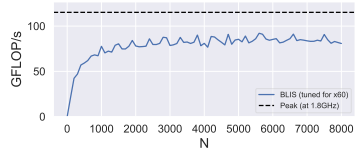
BLAS BENCHMARKS: DGEMM



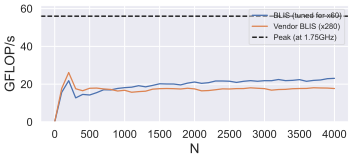
X60



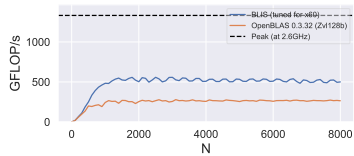
X100



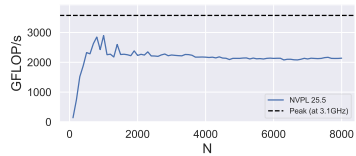
A100



X280

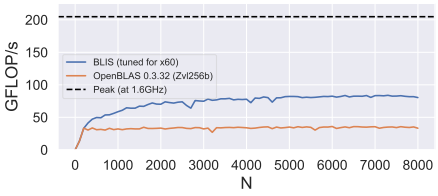


SG2044

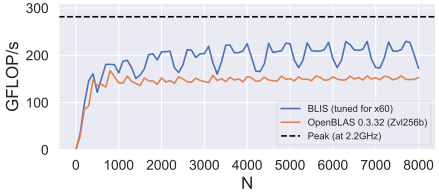


Grace

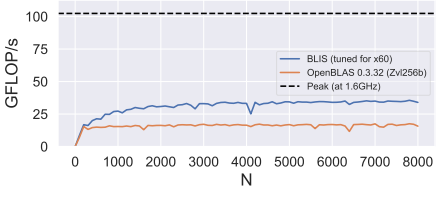
GENERATIONAL IMPROVEMENT



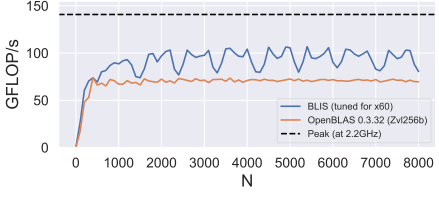
X60



X100



X60



X100

FFTW: INTRO

1D Complex-to-complex FFT

$$X[k] = \sum_{n=0}^{N-1} x[n]\, e^{-j\frac{2\pi}{N}kn}, \qquad k = 0, \ldots, N-1$$

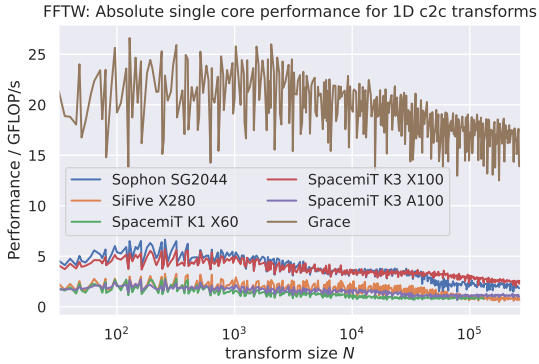
Where,

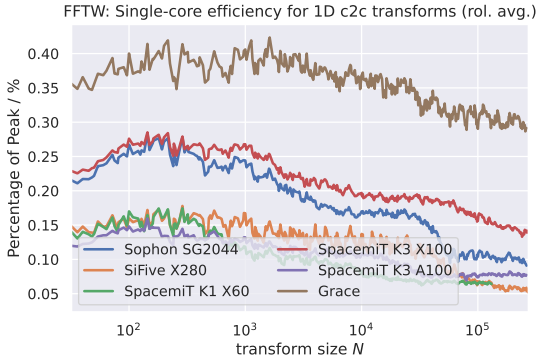
- N Transform size
- k Frequency-bin index
- $x, X \in \mathbb{C}^N$

Complexity scaling:

- $\mathcal{O}(N \log N)$ in time
- $\mathcal{O}(N)$ in core memory
- $\mathcal{O}(N)$ in plan/auxiliary memory

FFTW: RESULTS





- Performance gap with Grace cannot be explained away with just platform maturity
- Generational improvement in memory capabilities look promising for future hardware iterations

HPL

About

- A benchmark for measuring floating-point computing performance
- Solves a dense linear system using Gaussian elimination with partial pivoting.

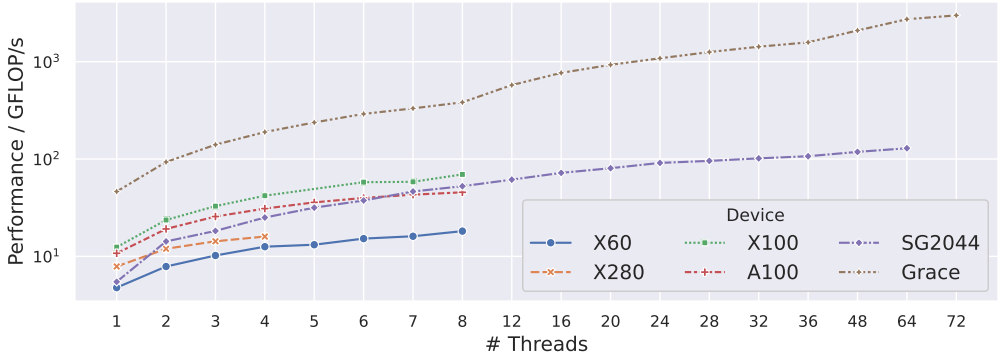
What it uses

- Dense matrix operations
- Highly optimized BLAS/LAPACKkernels
- Communication-aware parallelization

What it stresses

- Compute throughput
- Memory system performance
- Thread-level scalability

HPL: RESULTS



- RISC-V ecosystem is still far behind commercial systems
- Significant improvements needed to break through the 1 TFLOP/s barrier

HPCG: INTRO

About

- A benchmark for measuring realistic sparse computing performance
- Solves a sparse linear system using iterative methods with sparse matrix operations.

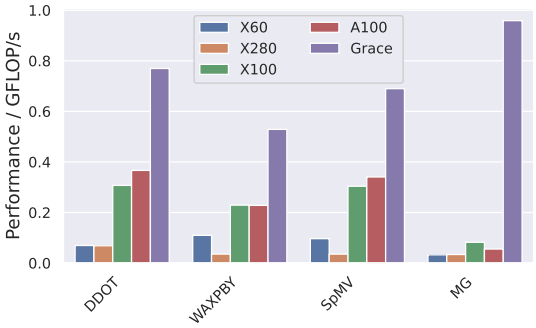
What it uses

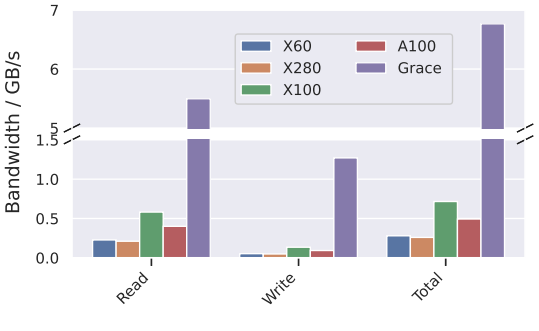
- Sparse matrix–vector multiplication (SpMV)
- Structured sparse computations (stencil/grid patterns)
- Multithreaded memory parallelization techniques

What it stresses

- Memory subsystem performance
- Thread-level scalability
- Contention effects

HPCG:RESULTS





- Grace: RISC-V has a lot of catching up to do – memory issues affect HPCG performance very severely.

SUMMARY OF RESULTS

- Benchmarks expose physical limits of hardware and microarchitectural bottlenecks
- BLAS benchmarks: Most platforms yield compute efficiencies between 30 % and 50 %. The exception is the SpacemiT K3, which is able to reach 80 %.
- FFTW benchmarks: Sophon SG2044 and SpacemiT X100 cores show a distinct advantage due to Out-of-Order execution.
- Comparisons to NVIDIA's ARM-based Grace CPU reveal that a substantial performance gap still remains.

CONCLUSIONS

- The rapid intergenerational improvements observed across the evaluated RISC-V designs strongly indicate that this performance gap is poised to close.
- Resolving the remaining readiness gaps requires both aggressively scaling out core counts, while simultaneously eliminating internal pipeline and bandwidth bottlenecks.
- The DARE project directly targets these bottlenecks by co-designing hardware and software to accelerate multithreaded, high-throughput applications on RISC-V.

CONCLUSIONS

- The rapid intergenerational improvements observed across the evaluated RISC-V designs strongly indicate that this performance gap is poised to close.
- Resolving the remaining readiness gaps requires both aggressiv core counts, while simultaneously eliminating i ecks.
- The DARE project directly targets these b and software to accelerate multithreaded, high applications on RISC-V.

Thank you for your attention!

p.chawla@fz-juelich.de

ACKNOWLEGEMENTS

The authors would like to thank:

- Funding for parts of this work has been received from EuroHPC's project DARE SGA 1 under Grant Agreement No. 101202459.
- The Monte Cimone project for providing access to the Sophon compute nodes used in this work.

INTERESTED IN FURTHER DISCUSSIONS? COME AND VISIT US!

Foyer | #Z02

Hall | #K02

PLEASE FOLLOW

juelich supercomputing centre @fzj-jsc.bsky.social @fzj_jsc @fzj_jsc

Upcoming JSC-Events: go.fzj.de/isc26