TEST-DRIVING RISC-V VECTOR HARDWARE FOR HPC

Joseph K. L. Lee, Maurice Jamieson, Nick Brown, Ricardo Jesus

epcc

Outline

- EPCC RISC-V Testbed
- RISC-V Vector Extension (RVV)
- Hardware Implementation
- Software Support: Compiler Toolchain, Linux, and Libraries
- Vector Benchmarks
- Summary & Recommendations

EPCC RISC-V Testbed

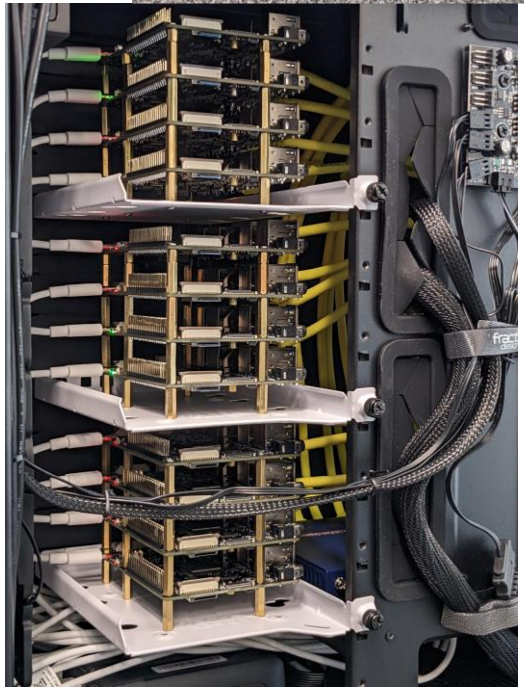
- Aim: Provide HPC code developers and data-scientists access to the latest RISC-V CPUs

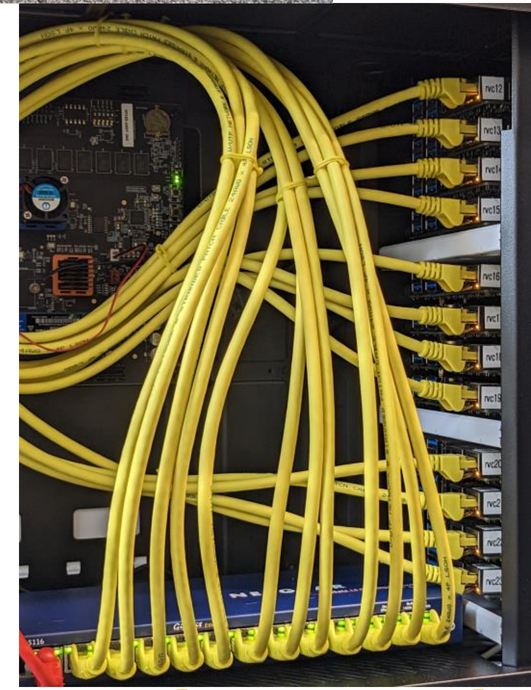
- We have many boards (64 cores):

Board	Processor (SOC)	# Core	Qty
NezhaSTU	C906 (D1)	1	4
MangoPi MQ-Pro	C906 (D1)	1	2
HiFive Unmatched	U74 (FU740)	4	1
StarFive VisionFive V1	U74 (JH7100)	2	3
StarFive VisionFive V2	U74 (JH7110)	4	13

- Also will have soft-cores
- Also have posts about building experiences
- Apply for access: http://riscv.epcc.ed.ac.uk/

- Funded by ExCALIBUR H&ES





RISC-V Vector Extension (RVV)

- Key feature of RISC-V: modular extensions
- Vector instructions useful in HPC applications:
  - exploit data parallelism, increase instruction bandwidth, improve energy efficiency
- Vector Extension (RVV) first proposed in 2015
- Important releases:
  - Version 0.7 (2019): stable enough to begin developing toolchains, simulators, implementations
    - (adopted by hardware e.g. C906, Vitruvius etc.)
  - Version 1.0: ratified in late 2021

RISC-V Vector Extension (RVV)

- Key features:
  - Vector length agnostic (VLA – like Arm SVE)
    - c.f. Vector Length Specific (VLS – like AVX, Arm NEON)
  - Vector length (VLEN) minimum 128 bits, up to 65,536 bits (c.f. Arm SVE max 2,048 bits)
  - Vector register grouping (LMUL): 1/2/4/8 registers
  - Fractional LMUL (not in RVV 0.7)

- Also a SIMD 'P' extension, aimed at embedded cores, low power DSP
  - Not yet ratified

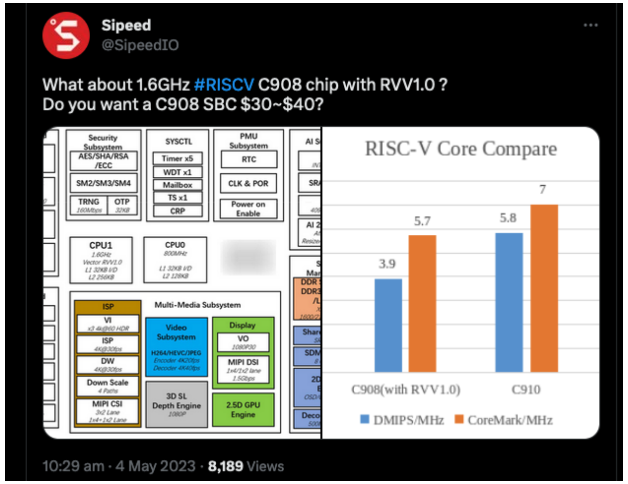
Vector Hardware Implementation

Processor	Vector Length (bits)	RVV version
SiFive P270/P470/P670	256 / 128 / dual-128	1.0
SiFive X280	512	1.0
Andes NX27V	Configurable from 128-512	1.0
Andes AX45MPV	Configurable from 128-1024	1.0
Vitruvius+	16384	0.7.1 (update to 1.0 in future)
Hwacha (V4)	512	Custom
New Ara	Configurable, e.g. 4096	1.0
Tenstorrent BOOM-ocelot	Configurable from 128	1.0
T-Head XuanTie C906/C920	128	0.7.1
T-Head XuanTie C908*	Configurable 128/256	1.0

epcc

Vector Hardware

- Wide range of applications:
  - General applications (SiFive P series)
  - Decoupled vector accelerator
    - Ara
    - Vitruvius+ : long vectors – 256 DP elements per register
- Off-the-shelf RVV 0.7:
  - T-Head (Alibaba) XuanTie C906
  - Found in Allwinner D1 SoC
  - 128-bit VLEN, support 8, 16, 32 bit vector elements
  - Does not support 64 bit elements(*), not suitable for HPC applications
- RVV 1.0?
  - XuanTie C908 with Sipeed?
- Softcores:
  - Some open source softcores: e.g. OpenC906, Tenstorrent Boom-ocelot
  - Requires knowledge for FPGA designs and tools



Vector Software Support: Compiler Toolchain

GNU

- Upstream GNU toolchain does not support vector extension
- rvv-next branch – limited support for RVV 1.0
- Older deleted branch rvv-0.7.1 (compiled mirror on EPCC website)
- T-Head provides modified GNU toolchain targeting C906
  - GCC 8.4 – Good auto-vectorisation (RVV 0.7)
  - GCC 10.2 – Intrinsics support, poor auto-vectorisation (RVV 0.7 & 1.0)
  - Mirror on EPCC website

LLVM

- LLVM 15 and 16 support RVV v1.0
- Support vector length agnostic (--scalable-vectorization=on) or vector length specific (--riscv-v-vector-bits-min/max=N)
- Support standard extensions with minimum vector length Zvl*, and embedded processors Zve*
- Results shown in upcoming talk: Backporting RISC-V vector assembly

Vector Software Support: Linux and Perf

Linux Kernel

- RISC-V Linux distribution generally available: Debian, Ubuntu, Fedora …
- Sipeed Linux image for Allwinner D1 supports RVV out of the box
  - However, bootloader is proprietary and protected, to modify Linux images must cross compile on another host, and vendor-specific patches must be applied to buildroot
  - Specific T-Head GCC compiler version must be used to ensure resulting image is RVV compatible
  - Time consuming & requires specific knowledge: high barrier to entry!

Performance Analysis & Instrumentation: Perf?

- To obtain events, kernel and OpenSBI need to be patched, depend on board & vendor
- HiFive Unmatched: the Linux kernel version 5.18 supports instruction and cycle count hardware events for perf
- Allwinner D1: official support for perf only released in Linux kernel version 6.2 on 19 Feb 2023, almost two years after the hardware was made available
- Major drawback for HPC workloads, where performance monitoring is necessary

Vector Software Support: Emulation and Libraries

Emulation

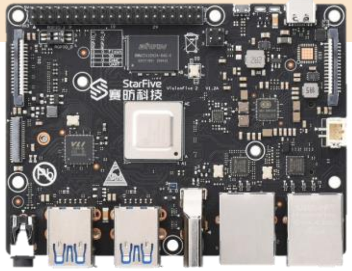
- Limited physical hardware, none yet for RVV 1.0
- QEMU, Spike: supports RVV 1.0 (earlier versions support RVV 0.7.1)
- Vehave (BSC):
  - Functional emulator based on QEMU
  - Dynamically handle and emulate vector instructions
  - Separate versions supporting RVV 1.0 and 0.7.1

Libraries

- Most HPC libraries can be cross-compiled for RISC-V, but tend to have limited vectorisation optimisation
- OpenBLAS optimised for RVV 0.7.1, requires specific compiler from T-Head (v2.6.0 toolchain)
- Effort within community to optimise libraries (e.g. FFTW)
- Likely see significantly increased support within the next year

Vector Benchmarks: Systems

	Allwinner D1	StarFive JH7110 (VF2)	A64FX
Processor	XuanTie C906	SiFive U74	Fujitsu A64FX
Clock Speed	1.0 GHz	1.5 GHz	1.8 GHz
Cores	1	4	48
Cache	32 KB I-cache + 32 KB D-cache	32 KB I-cache + 32 KB D-cache + 2MB L2	64 KB I-cache + 64KB D-cache, 8 MB shared L2 per 12 cores
Memory	512MB DDR3	8GB DDR4	32GB HBM2
ISA	RV64GC+V0.7	RV64GC	ARMv8.2 with SVE
Vector width	128 bit	N/A	Dual 128-bit (NEON)/ Dual 512-bit (SVE)



epcc

Vector Benchmark

- Only on single core
- Single precision
- For A64FX, use NEON only
  - 128-bit vector length, same as D1
  - T-Head compiler generates VLS code (fixed 128-bit)
- A64FX is designed for HPC vs RISC-V cores for embedded / single-board computer:
  - Still interesting to compare

Vector Benchmark

- Benchmark: RAJA Performance Suite (https://github.com/LLNL/RAJAPerf)
  - ALGORITHM
  - APPS
  - BASIC
  - LCALS (Livermore Compiler Analysis Loop Suite)
  - POLYBENCH
  - STREAM

Name	Compiler	Vector width	Compiler flags
RV-GCC8.4-scalar	XuanTie GCC8.4	N/A	-O3 -march=rv64gc -ffastmath
RV-GCC8.4-vector	XuanTie GCC8.4	128-bit	-O3 -march=rv64gcv -ffastmath
ARM-GCC11.2-scalar	GCC 11.2	N/A	-O3 –ffastmath -mcpu=a64fx -march=armv8.2-a+nosimd+nosve
ARM-GCC11.2-vector	GCC 11.2	128-bit	-O3 –ffastmath -mcpu=a64fx -march=armv8.2-a+simd+nosve

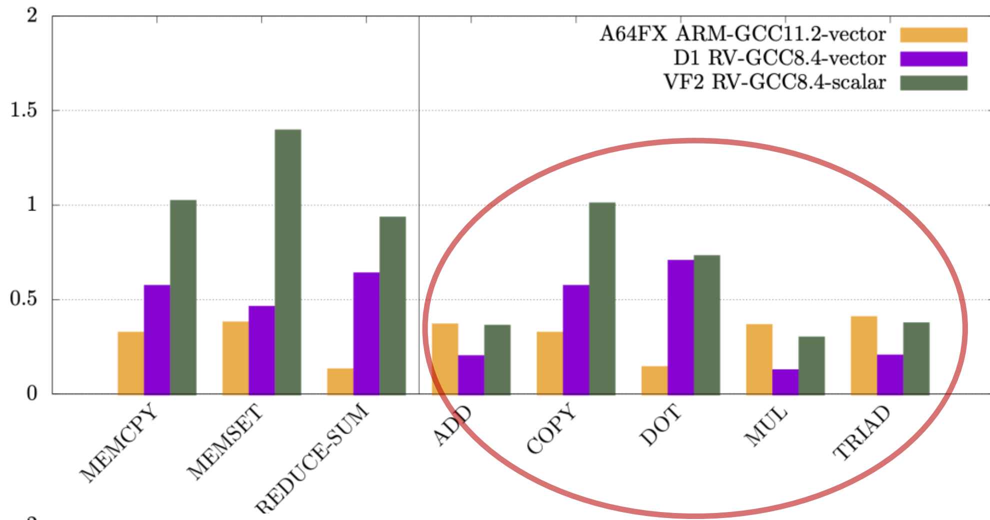
Vector Benchmark: Results

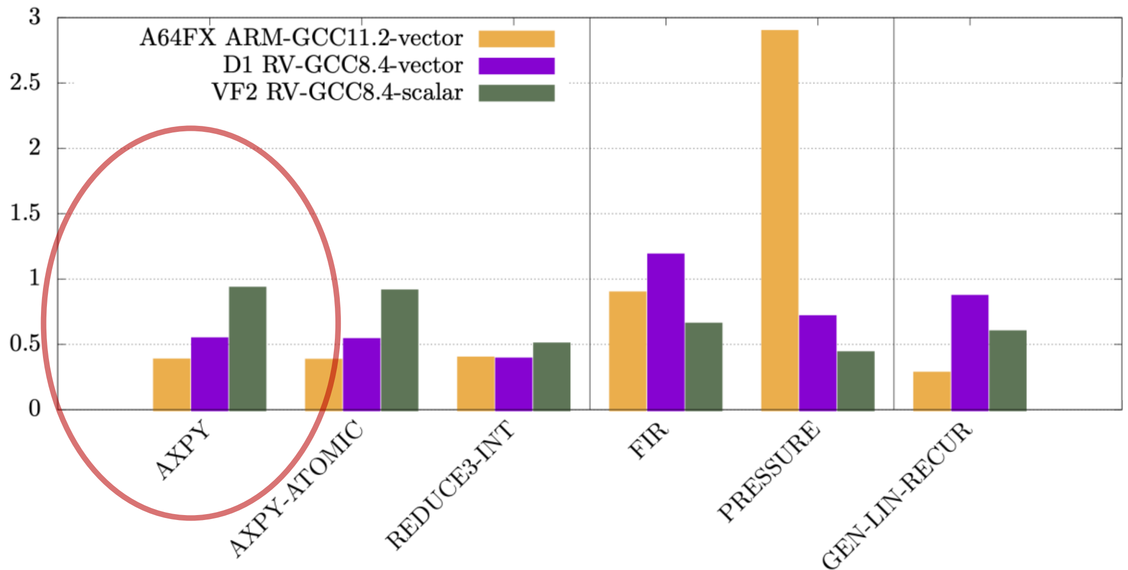
- For RV-GCC8.4-vector, out of 64 kernels:
  - 23 vectorised and vector loop executed
  - 7 vectorised but vector loop not executed
  - 34 only scalar
  - Clang vectorises more kernel than GCC (See next talk)
  - Vectorised kernel sensitive to loop ranges, scalar branch taken often

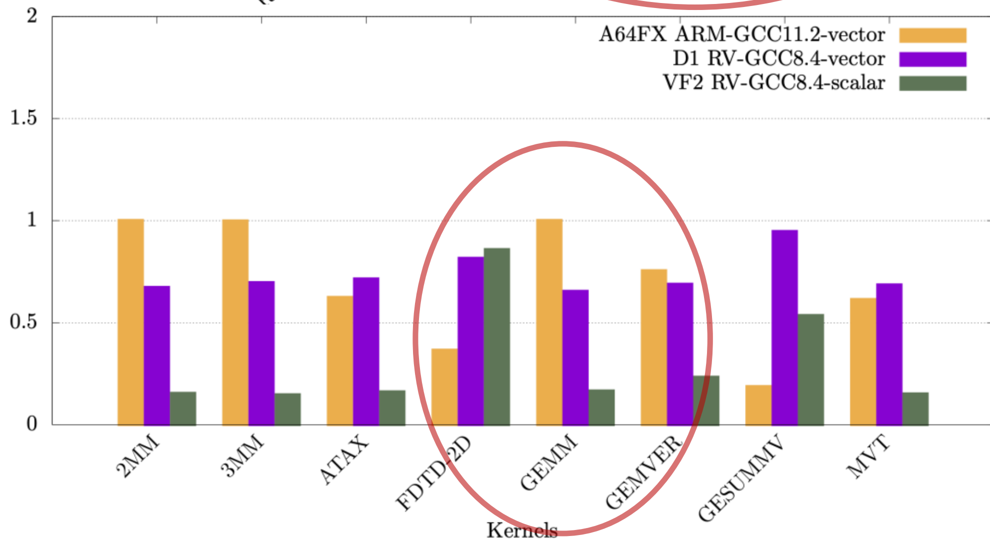
Vector Benchmark: Results

- Summary:
  - Purple: D1-vector / D1-scalar
  - RVV achieves higher bandwidth for stream kernels
  - RVV accelerates Linear Algebra kernels:
    - 84% faster for AXPY
    - 53% faster for GEMM…
  - Speedup generally not as significant as NEON on A64FX

RISC-V timing normalised against D1 scalar
A64FX vector timing normalised against A64FX scalar
Lower is better



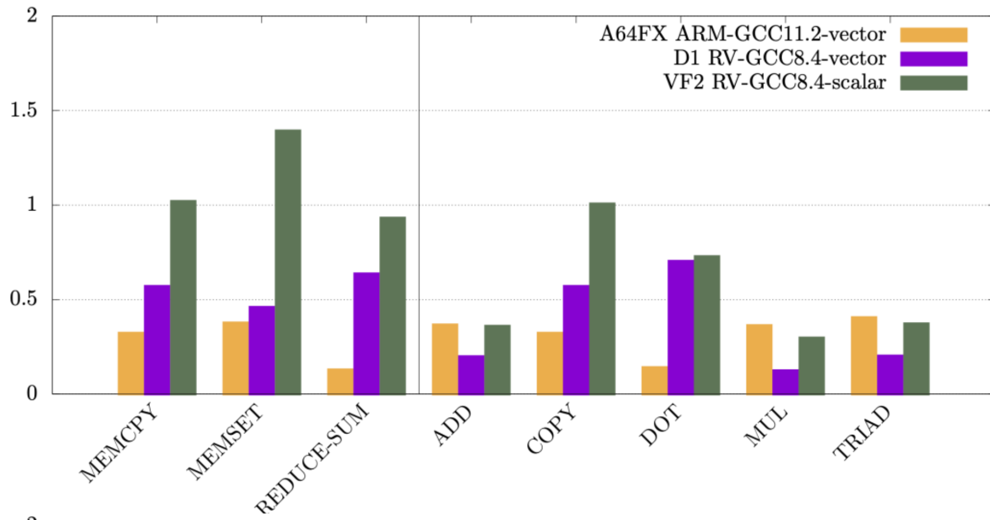


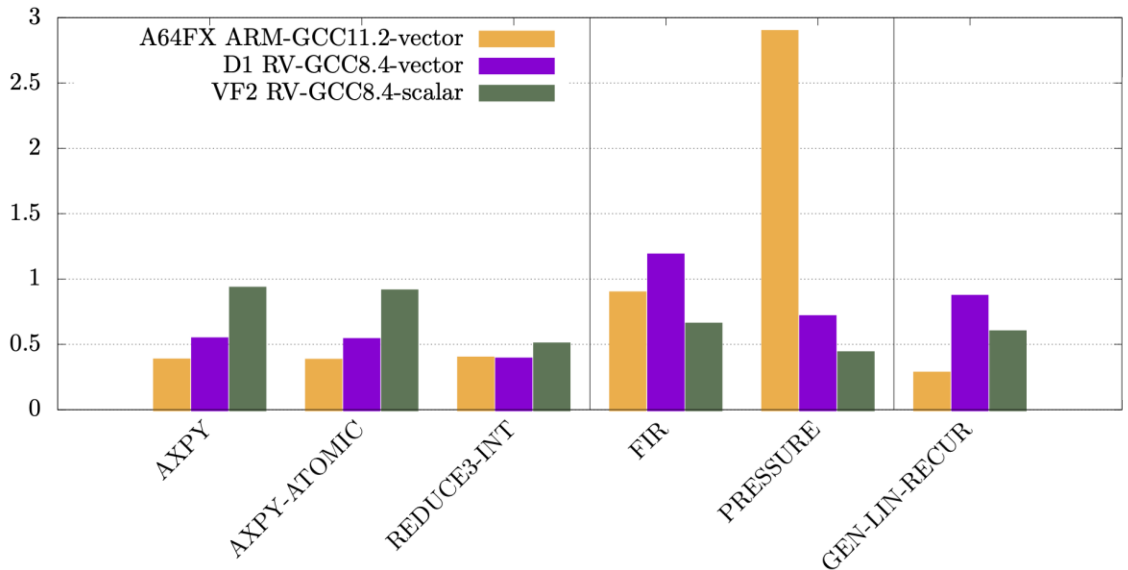


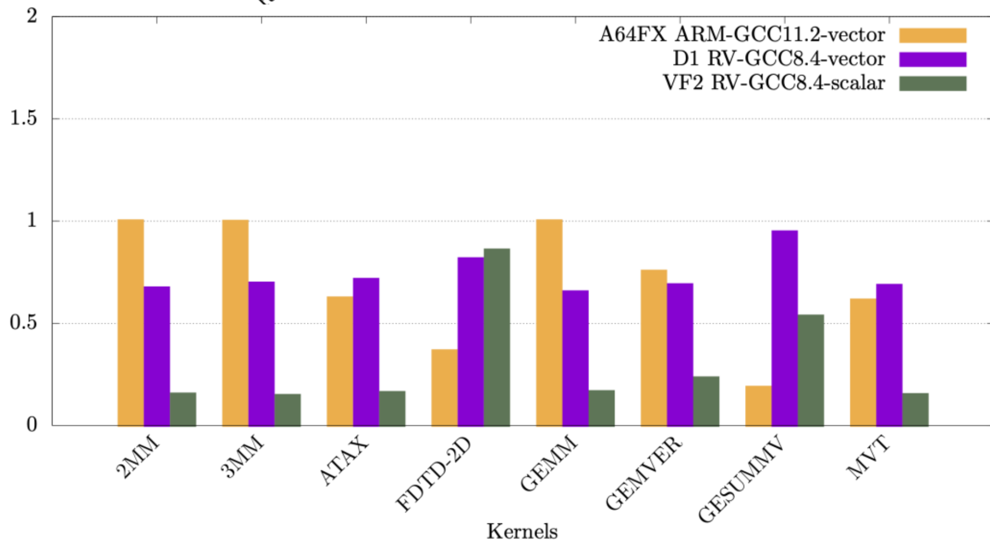
Vector Benchmark: Results

- Allwinner D1 vs StarFive JH7110 (VF2) (Green):
  - VF2 higher frequency, GEMM 6x faster than D1 scalar, 4x D1 w/ vector
  - But with vectorisation D1 streaming is faster than VF2, even though VF2 has higher theoretical bandwidth
  - AXPY on D1 w/ vector 77% faster than VF2 scalar
  - D1 considerably cheaper than VF2, impressive
  - But only testing 1 out of 4 cores in VF2

RISC-V timing normalised against D1 scalar
A64FX vector timing normalised against A64FX scalar
Lower is better







Summary

- D1 gains significant performance advantage with RISC-V Vector extension
- Mismatch between RVV version in available tooling (e.g. GCC and Clang) and hardware makes running and testing RVV codes difficult
- Challenges due to immaturity will hopefully be solved with standardisation of tooling and RVV 1.0 compliant hardware
- RVV provides a strong foundation for leveraging RISC-V for high performance workloads
- Improvement potentials to further increase performance:
  - improved auto-vectorisation in LLVM
  - increased VLEN in future CPUs
- Would be helpful if support present for both RVV 0.7 and 1.0 in mainstream GCC and Clang

Recommendations

- We recommend using the T-Head GCC 8.4 auto-vectorisation and *not* using the T-Head RVV v0.7 intrinsic API

- This ensures that codes can simply be recompiled, without modification, to target RVV v1.0 compatible hardware

- We also recommend building RVV-enabled Linux images with a patched mainstream buildroot using the T-Head GCC 8.4 compiler, as support for the Allwinner D1 has recently been added

Thank you!

- Next part: Backporting RISC-V Vector assembly

- EPCC RISC-V Testbed: http://riscv.epcc.ed.ac.uk/

Additional slides: 1

- RAJAPerf kernels vectorised by RV-GCC8.4-vector

Vectorised and executed: Total 23	
Algorithm	MEMCPY, MEMSET, REDUCE_SUM
Apps	ENERGY, FIR, PRESSURE
Basic	AXPY, AXPY_ATOMIC, REDUCE3_INT
Lcals	GEN_LIN_RECUR
Polybench	2MM, 3MM, ATAX, FDTD 2D, GEMM, GEMVER, GESUMMV, MVT
Stream	ADD, COPY, DOT, MUL, TRIAD

Vectorised: Total 7	
Lcals	FIRST_SUM, FIRST_DIFF, HYDRO_1D, HYDRO_2D, TRIDIAG_ELIM
Polybench	JACOBI_1D, JACOBI_2D

Scalar: Total 34	
Algorithm	SCAN, SORT, SORTPAIRS
Apps	CONVECTION3DPA, DEL_DOT_VEC_2D, DIFFUSION3DPA, HALOEXCHANGE, HALOEXCHANGE_FUSED, LTIMES, LTIMES_NOVIEW, MASS3DPA, NODAL_ACCUMULATION_3D, VOL3D
Basic	IF_QUAD, INDEXLIST, INDEXLIST_3LOOP, INIT_VIEW1D, INIT_VIEW1D_OFFSET, INIT3, MAT_MAT_SHARED, MULADDSUB, NESTED_INIT, PI_ATOMIC, PI_REDUCE, REDUCE_STRUCT, TRAP_INT
Lcals	DIFF_PREDICT, EOS, FIRST_MIN, INT_PREDICT, PLANCKIAN
Polybench	ADI, FLOYD WARSHALL, HEAT_3D