

Accelerating Unicode Conversions using the RISC-V Vector Extension

Olaf Bernstein

Motivation

- The majority of digital text is stored in UTF-8 format
- Some languages and APIs use other formats internally, e.g. UTF-16 is used by JavaScript, Java and Windows
- Conversion can be optimized by processing multiple characters in parallel, leading to large speedups

RISC-V Vector Extension

- Vector length agnostic
- Tail and mask predicated instructions
- Powerful compress and gather permutations
- Very orthogonal instruction set
- LMUL register grouping

simdutf library[1]

- Vectorized Unicode conversions for different ISAs: SSE, AVX, AVX2, AVX512, NEON, **now: RVV**
- Used in popular libraries: Node.js and Bun

RVV simdutf backend

- Vectorized all routines*
- About 1K lines of code (1/3 of custom AVX512 backend)
- Optional Zvbb for endianness reversal

*We vectorized all routines that were supported as of March 2024. That is, all conversions between UTF-8, UTF-16, UTF-32, latin1, as well as the respective validation and length calculations. Newer versions of simdutf include base64 conversions, which haven't been added to the RVV backend yet. We've already explored a few implementation strategies, however, the current limited hardware variety made it hard to decide which one would perform best on the majority of future hardware.

Performance Exploration

Alongside this work, we developed a collection of RVV benchmarks to help developers write performance portable RVV code. [2]

Optimization Techniques

- fast paths
- unroll LMUL=1 `vrgather.vv`, when not crossing lanes
- avoid LMUL=8 `vcompress.vv`, due to current hardware
- otherwise maximize LMUL without spilling, which amortizes scalar and mask instructions

UTF-8 to UTF-16 conversion

- Validate UTF-8 using 3 4-bit LUTs [3]
- Identify character positions
- Extract all leading UTF-8 bytes and the three following bytes into four separate vector registers
- Remove prefixes
- Combine to UTF-32 code point
- Convert code points >0xFFFF to surrogate pairs
- Remove upper 16-bit of code points ≤0xFFFF

Test Hardware

	C908	C920	X60
Vendor	XuanTie	XuanTie	SpacemiT
Vector extension	RVV 1.0	XTheadVector	RVV 1.0
scalar ISA	RV64GCB	RV64GC	RV64GCB
VLEN	128	128	256
Dispatch	2x64	2x128	2x128
Execution model	in-order	out-of-order	in-order

Note: Our simdutf backend only supports XTheadVector using GCC's capability of compiling RVV intrinsics to XTheadVector, but the benchmarks include measurements from manually ported assembly.

Visualized conversion of "rvv👷"

character	r	v	v	v			👷			
UTF-8	01110010	11001110	10111101	11100001	10111001	10111111	11110000	10011111	10100111	10011001
identify	01110010	11001110	10111101	11100001	10111001	10111111	11110000	10011111	10100111	10011001
byte 1	01110010	11001110	11100001	11110000		01110010	00001110	00000001	00000000	
byte 2	11001110	10111101	10111001	10011111	remove	00001110	00111101	00111001	00011111	
byte 3	10111101	11100001	10111111	10100111	prefixes	00111101	00100001	00111111	00100111	
byte 4	11100001	10111001	11110000	10011001		00100001	00111001	00110000	00011001	
widen and combine	00000000	11100110	00111011	10000111	shift to	00000000	00000000	00000000	00000000	11100110
	00000000	00011100	11111101	110000	UTF-32	00000000	00000000	00011100	11111101	
	00000000	00001111	10011011	1011001		00000000	00001111	10011011	1011001	

Table 1: UTF-8 to UTF-32 conversion visualized

Performance Improvements

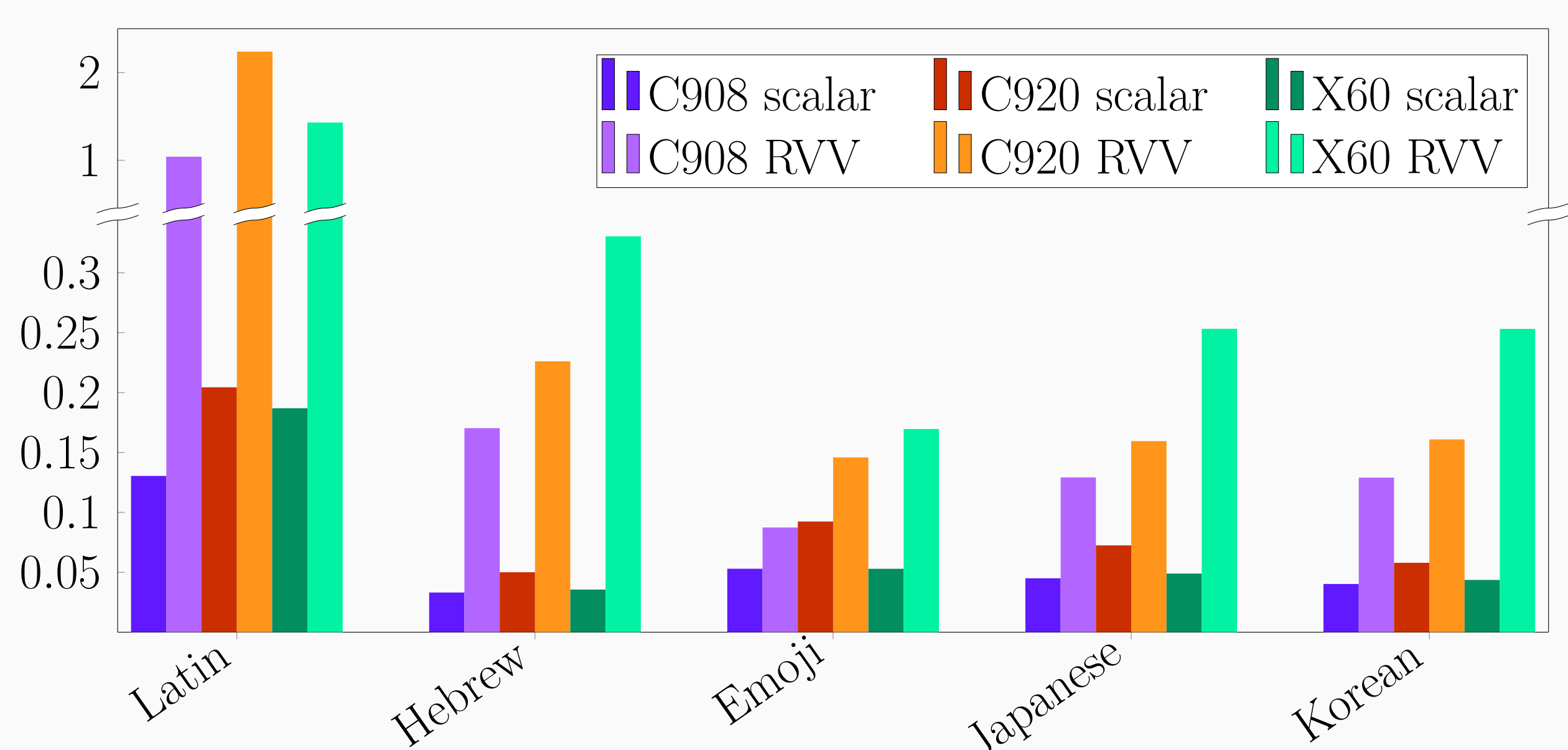


Figure 1: UTF-8 to UTF-16 (input bytes/cycle)

Comparison with processors of other vector ISAs

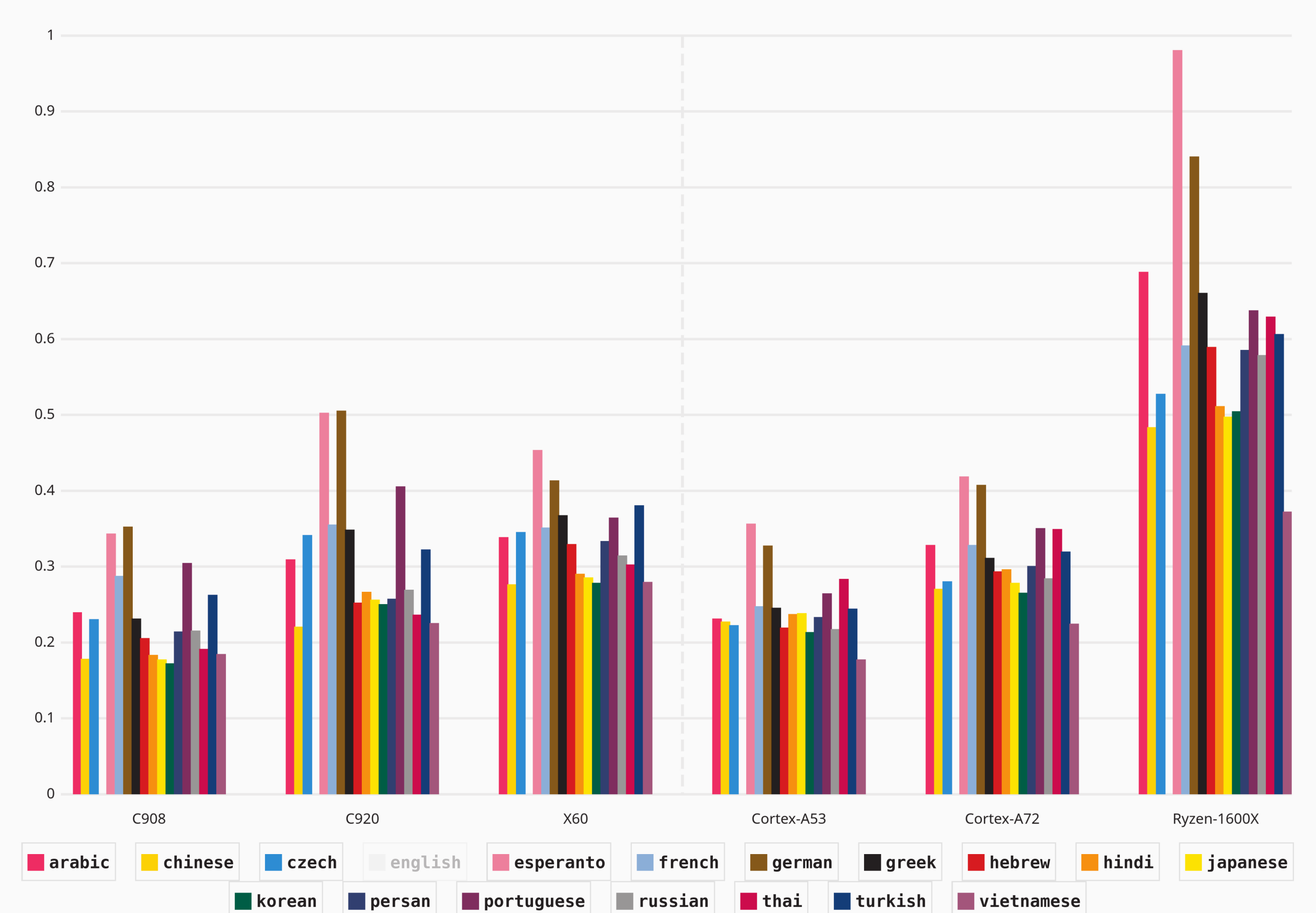


Figure 2: UTF-8 to UTF-16 (input bytes/cycle)




Conclusion

We've managed to effectively use the RISC-V Vector extension to speed up UTF-8 to UTF-16 conversion by on average 3-5 times on real hardware. This and the other supported text conversion functions have been upstreamed to the simdutf library and can now benefit software using simdutf, e.g. software based on Node.js and Bun.

Future work

- Vectorize the base64 routines added in the latest version of simdutf
- Benchmark on more implementations: SiFive-P670, XiangShan, ...
- Investigate possible `vlut4.vv/vrgatherei4.vv` instruction

References

- [1] Daniel Lemire et al. "simdutf"  [simdutf/simdutf](#)
- [2] Olaf Bernstein "rvv-bench"  [camel-cdr/rvv-bench](#)
- [3] John Keiser and Daniel Lemire. "Validating UTF-8 In Less Than One Instruction Per Byte" (2020)  [arXiv: abs/2010.03090](#)

More detail

