Check for updates

BRIEF REPORT

REVISED Co-designing ab initio electronic structure methods on a RISC-V vector architecture

[version 4; peer review: 5 approved]

Rogeli Grima Torres, Pablo Vizcaíno, Filippo Mantovani, José Julio Gutiérrez Moreno

Barcelona Supercomputing Center (BSC), Plaça Eusebi Güell, 1-3, Barcelona, 08034, Spain

v4 **First published:** 05 Aug 2024, **4**:165
https://doi.org/10.12688/openreseurope.18321.1
Second version: 28 Oct 2024, **4**:165
https://doi.org/10.12688/openreseurope.18321.2
Third version: 14 Nov 2024, **4**:165
https://doi.org/10.12688/openreseurope.18321.3
Latest published: 09 Sep 2025, **4**:165
https://doi.org/10.12688/openreseurope.18321.4

Abstract

Ab initio electronic structure applications are among the most widely used in High-Performance Computing (HPC), and the eigenvalue problem is often their main computational bottleneck. This article presents our initial efforts in porting these codes to a RISC-V prototype platform leveraging a wide Vector Processing Unit (VPU). Our software tester is based on a mini-app extracted from the ELPA eigensolver library. The user-space emulator Vehave and a RISC-V vector architecture implemented on an FPGA were tested. Metrics from both systems and different vectorisation strategies were extracted, ranging from the simplest and most portable one (using autovectorisation and assisting this by fusing loops in the code) to the more complex one (using intrinsics). We observed a progressive reduction in the number of vectorised instructions, executed instructions and computing cycles with the different methodologies, which will lead to a substantial speed-up in the calculations. The obtained outcomes are crucial in advancing the porting of computational materials and molecular science codes to (post)-exascale architectures using RISC-V-based technologies fully developed within the EU. Our evaluation also provides valuable feedback for hardware designers, engineers and compiler developers, making this use case pivotal for co-design efforts.

Keywords

High-performance computing, co-design, ab initio, RISC-V, materials science, eigensolver library, European processor initiative

Open Peer Review

Approval Status ✓ ✓ ✓ ✓ ✓

	1	2	3	4	5
version 4 (revision) 09 Sep 2025					✓ view
version 3 (revision) 14 Nov 2024					? view
version 2 (revision) 28 Oct 2024				✓ view	
version 1 05 Aug 2024	✓ view	✓ view	✓ view	? view	

1. **Flavio Vella**, University of Trento, Trento, Italy
2. **Ahmed Kamaleldin**, Technische Universitat Dresden, Dresden, Germany
3. **Roman Wyrzykowski**, Czestochowa University of Technology, Częstochowa, Poland
4. **William Dawson**, RIKEN Center for Computational Science, Kobe, Japan
5. **Keichi Takahashi**, University of Osaka,



This article is included in the Horizon Europe gateway.



This article is included in the Horizon 2020 gateway.

Osaka, Japan

Any reports and responses or comments on the article can be found at the end of the article.

Corresponding author: José Julio Gutiérrez Moreno (julio.gutierrez@bsc.es)

Author roles: Grima Torres R: Conceptualization, Data Curation, Formal Analysis, Investigation, Methodology, Project Administration, Software, Validation, Visualization, Writing – Original Draft Preparation, Writing – Review & Editing; **Vizcaíno P**: Conceptualization, Formal Analysis, Investigation, Methodology, Validation, Writing – Review & Editing; **Mantovani F**: Conceptualization, Formal Analysis, Investigation, Methodology, Resources, Supervision, Validation, Writing – Review & Editing; **Gutiérrez Moreno JJ**: Conceptualization, Formal Analysis, Funding Acquisition, Investigation, Methodology, Project Administration, Resources, Supervision, Writing – Original Draft Preparation, Writing – Review & Editing

Competing interests: No competing interests were disclosed.

Grant information: This work has been developed within the NOMAD Centre of Excellence (CoE), funded by the European Union's Horizon 2020 research and innovation program under grant agreement No. 951786. We also acknowledge the European Union's MaX CoE, under grant agreement No. 101093374, co-funded by the European HPC Joint Undertaking (JU) and participating countries, and the Machine-learning-driven bottom-up design of atomically-layered heterostructures for green production (MLALH) project, awarded by the EIG CONCERT-Japan call with reference PCI2023-143426 funded by MCIN/AEI/10.13039/501100011033 and the European Union. The EPAC development is supported by the EuroHPC Joint Undertaking (JU): FPA No. 800928 (EPI), SGA No. 101036168 (EPI-SGA2). The JU receives support from the EU Horizon 2020 research and innovation programme and from Croatia, France, Germany, Greece, Italy, Netherlands, Portugal, Spain, Sweden, Denmark and Switzerland. The EPI-SGA2 project, PCI2022-132935 is also co-funded by MCIN/AEI /10.13039/501100011033 and by the UE NextGenerationEU/PRTR.
The funders had no role in study design, data collection and analysis, decision to publish, or preparation of the manuscript.

Copyright: © 2025 Grima Torres R *et al*. This is an open access article distributed under the terms of the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

How to cite this article: Grima Torres R, Vizcaíno P, Mantovani F and Gutiérrez Moreno JJ. **Co-designing ab initio electronic structure methods on a RISC-V vector architecture [version 4; peer review: 5 approved]** Open Research Europe 2025, **4**:165 https://doi.org/10.12688/openreseurope.18321.4

First published: 05 Aug 2024, **4**:165 https://doi.org/10.12688/openreseurope.18321.1

REVISED **Amendments from Version 3**

Conclusions section updated.

Any further responses from the reviewers can be found at the end of the article

Introduction

Ab initio electronic structure applications are among the most popular and computationally demanding in High-Performance Computing (HPC). In recent years, developers have put substantial efforts into modularising codes, moving from rather extensive and sometimes monolithic applications toward more structured software. This new design intends to be more adaptable to the incorporation of external libraries for some critical or computationally expensive parts of the codes[1]. This adaptation will facilitate the evolution of codes to the (post-)exascale era and to perform efficiently on heterogeneous computing systems[2].

Eigenvalue problems are key in *ab initio* electronic structure calculations when solving the Schrödinger equation for many-body extended systems. Eigensolvers are often the main computational bottleneck in Density Functional Theory (DFT) calculations. This is due to the cubic scaling growth in computational costs with respect with the problem size, which limits the size and complexity of the model in practice. From our experience as users and developers, and based on extensive performance analyses, we have observed that eigenvalue solvers can consume up to over 90% of the total computational time for relatively large systems. The ELPA (Eigenvalue soLvers for Petaflop Applications) library is designed to solve dense eigenvalue problems (e.g. matrices where most or all elements are non-zero) efficiently, while it is also used as a building block for sparse solvers, supporting efficient CPU and GPU performance on all major HPC platforms[3–5]. ELPA is employed by some of the most widely-used DFT codes, such as VASP[6], Siesta[7], Quantum Espresso[8], Abinit[9], exciting[10], FHI-aims[11], GPAW[12] and CP2K[13]. The library can be directly integrated within the code or incorporated as a part of wider open-source software library solutions, such as ELSI[14] or SIRIUS[15].

Exascale computing represents a significant advancement in HPC, unlocking unprecedented opportunities to transform materials and molecular modeling, allowing more accurate calculations, complex morphologies and exploration of large data sets to discover novel compounds[16]. However, this evolution will come with more heterogeneity in computer architectures, incorporating specialised hardware for specific applications[17]. Therefore, bidirectional efforts in co-design of hardware and software are required from early-developed prototype systems toward an efficient transition to the post-exascale era.

Our paper describes the initial steps to port the ELPA eigensolver to a prototype platform called EPAC-VEC powered by a RISC-V core coupled with a wide vector unit. This architecture is part of EPAC, a collection of RISC-V based accelerators implemented in a test chip fabricated during 2023 at 22nm within the European Processor Initiative (EPI) project. EPI seeks to reinforce Europe's digital sovereignty by developing high-performance, energy-efficient processors for supercomputers. EPAC-VEC is an extremely relevant architecture for general purpose HPC since it features a vector unit capable of handling vectors of up to 256 double-precision elements (16,384 bits per vector register)[18], 32 times larger than current Single Instruction/Multiple Data (SIMD) architectures such as Intel's AVX512 extension. This long-vector architecture has already been used to accelerate other HPC workloads such as Fast Fourier Transforms (FFT)[19], sparse matrix-vector multiplication[20] or graph processing algorithms[21]. We have also tested EPAC-VEC for other scientific software applications such as Alya[22], a computational fluid dynamic (CFD) code developed in Fortran and we demonstrated speed-ups up to 7× using code modifications that leave the code portable and deliver performance improvements also on Intel x86 architectures (while leveraging AVX512 SIMD extension by Intel). Given the relevance of eigensolvers, optimisations made on ELPA will eventually benefit the wider community and pave the way for the future portability of electronic structure applications to RISC-V architectures. On the other hand, as a cornerstone of the co-design process, the outcomes collected during pioneering works are also beneficial in guiding the development of future hardware and compilers.

Methodology

Our performance analyses were carried using the so-called Software Development Vehicles (SDV)[23]. This set of platforms, compilers, and analysis tools allow software developers to run applications on early iterations of the hardware, providing constant feedback to architecture design and the compiler development team, which guarantees the possibility of quickly improving the platform design.

The initial executions were performed on Vehave[23], a user-space emulator for the RISC-V Instruction Set Architecture (ISA) vector extension. Vehave runs on top of RISC-V commercial platforms, intercepting the vectorised instructions, decoding them, and emulating the vector extension. The emulator relies on LLVM[24] libraries for instruction decoding and generates detailed Paraver[25] trace files containing information about each emulated vector instruction. In addition to the Vehave platform, where vector instructions are emulated, we also used a field-programmable gate array (FPGA)-based emulation of the EPAC-VEC chip[23]. This FPGA is used as a user-defined reconfigurable hardware platform emulating the EPAC-VEC test chip. While emulators are useful for early testing and development, FPGAs enable us to test the design in real-world conditions with accurate timing, ensuring that the system meets performance targets in a hardware environment. The compiler, based on LLVM, supports autovectorisation and provides built-ins for vector instructions. A reference of the vector EPI intrinsics can be consulted online[26]. The Vehave emulator and the FPGA implementation serve different purposes and offer unique advantages. Vehave allows for quick iteration, debugging, and testing of different configurations without needing to reconfigure hardware. This is particularly useful when exploring architectural optimisations or validating functionalities under varying conditions. On the other hand, the FPGA implementation offers cycle-accurate performance and validates

that our design operates as expected when mapped onto a physical implementation.

Given the limitation of a single (emulated) computing core, the early-stage development of the compiler and the limited availability of libraries, this work has been performed using a mini-app extracted from ELPA, representing a small fraction of the code that retains the primary performance-intensive section. Our mini-app was inspired by a broader suite, developed in the NOMAD Center of Excellence[27] framework, to drive co-design in *ab initio* electronic structure calculations. More details on the mini-apps development and execution are given in our recent publication[28] and its associated repository (https://gitlab.bsc.es/material-science/nomad-mini-apps-suite). Our code isolates the *trans_ev_tridi_to_band* subroutine in ELPA (v.2022.05.001), extracted from the two-stage tridiagonalisation[3]. This method is normally preferred in large problems and when most eigenvectors must be computed. The kernel was selected based on its computational cost, independence from external functions (Basic Linear Algebra Subroutines (BLAS) library or communications do not dominate the function), and especially for the extensive effort the ELPA developers made to adapt this kernel to use vector instruction on different hardware efficiently (i.e., SSE, AVX(2/512), SPARC64 SSE, ARM SVE(128/256/512), BlueGene/(P/Q), NVIDIA, AMD and Intel GPUs).

All the code developed within this project, the instructions to execute the mini-app on the RISC-V environment and the ELPA checkpoints (for real double-precision random matrices using the 2-stage method solver with different sizes) are accessible from the associated online repositories. We saved different checkpoints of the application at the entrance of the function *trans_ev_tridi_to_band*. Each checkpoint is labeled with the matrix size, the block size, and the number of eigenvectors that are computed. Note that we have vectorised our code in the direction of the number of eigenvectors and it is very sensitive to this value. For the sake of correctness, we tested all the kernels of our code using Vehave, using every available checkpoint. To test the vector behavior, we focused on our most relevant cases, with large matrix sizes and a number of eigenvectors greater than the vector length. The default checkpoint file is a matrix of 1024 elements, 256 eigenvectors and a blocksize of 32 elements.

Results and discussion

Here, we describe the iterative steps toward adapting the ELPA mini-app to run efficiently on a long vector architecture. The ELPA kernels were initially converted from Fortran to C. The C version of the code allowed full compatibility with version 0.7.1 of the EPI LLVM compiler and the execution on the FPGA platform. Whereas the binaries produced from the two codes where different, the number and type of vector instructions produced by both versions were proven to be analogous. We should remark that at the time this work was carried out, Vehave used the latest v1.0 implementation of EPI, while the FPGA was on the previous v0.7.1. This new v1.0 of the V-extension allows the compilation of codes in both Fortran and C, while v0.7.1 did not allow Fortran compilations.

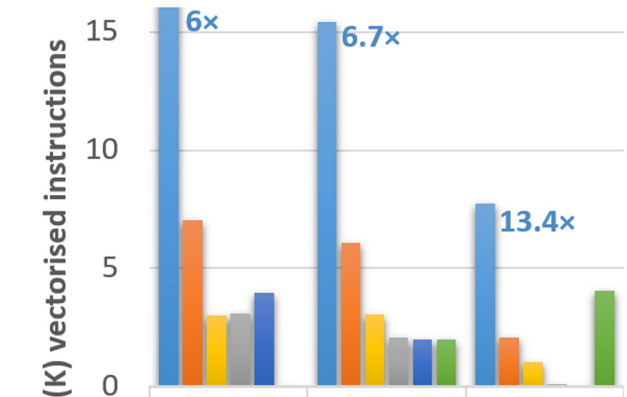
After that, the first step was compiling the scalar mini-app on a commercial RISC-V core. This was done to verify the code's compatibility with the RISC-V architecture, that the compiler supports all data structures and code features, and that the instructions are equivalent in both the C and Fortran versions.

Our initial testing on the VPU used the Vehave emulator. While this application does not give access to measuring computing cycles or execution time, the traces provide the number and type of each executed vector instruction, which is valuable insight for studying code regions with vectorisation potential. Based on that information, we have studied, analysed and enabled the vectorisation of ELPA kernels.

The vectorisation was achieved using three different strategies, starting with the least effort and gradually increasing in complexity. These are: (i) enabling compiler auto-vectorisation capabilities, (ii) helping the compiler fuse loops, and (iii) vectorising manually using intrinsics. This bottom-up approach offers several possibilities, starting from the simplest and most portable method, and evolving toward more complex and time-consuming processes from which a better performance is expected to be obtained.

By leveraging the compiler's autovectorisation capabilities in the ELPA mini-app's original version, Vehave performance analyses counted a total of 103,784 vector instructions. However, the traces show that most of the work is done with a vector length of 48 double-precision elements. This happens because ELPA was designed to divide its Q matrix into 48-element stripes. This implementation is very efficient for exploiting memory locality. However, long-vector architectures are more resistant to memory latency[29], and would rather benefit from a long vector length. Therefore, this striped distribution came out to be suboptimal for the VPU. Subsequently, by suppressing the Q-stripes, the algorithm starts leveraging its full vector length (256 elements per vector) without the limitation of 48 elements, and the vector instructions are reduced by a factor of 6 × compared to the original version (from 103,784 to 17,304 vector instructions). We should note that since our Vehave version was shipped with the v1.0 of the EPI compiler, we could verify that the outcomes from the Fortran and C versions were equivalent at this stage. The number of vector instructions for each code version is presented in Figure 1, where the different instruction types are noted in the caption.

Upon inspection of the compilation output, we still identify several loops in which the compiler is not able to reuse loaded vectors, so memory accesses are replicated in a sub-optimal manner. Therefore, our next strategy consisted of assisting the compiler by identifying loops going through the same variables and combining (fusing) them in the code. This strategy allows us to reduce our memory accesses (*vle* calls in Figure 1) from 7,040 to 6,064. With these changes, the total number of instructions improves by a factor of 6.7 ×, an additional 11% with respect to the previous version (from 17,304 to 15,405). Moreover, this strategy is expected to improve the performance of any vectorising compiler.



	auto	fused	intrinsics
TOTAL	17.304	15.405	7.728
vle	7.040	6.064	2.096
vse	3.024	3.040	1.040
vsetvli	3.107	2.099	0.112
vfmadd	3.968	2.000	0.016
vfmacc	0.016	2.000	4.048

Figure 1. Count of vectorised instructions in the different versions of the ELPA mini-app executed on Vehave. The numbers in the bar diagram indicate the proportional reduction in vector instructions compared to our initial version with suppressed Q-stripes. Instructions are distributed by total (light blue bars), 64-bit unit-stride load (vle, orange) and save (vse, orange), settings of vector length (vsetvli, gray), and multiplication-additions (vfmadd in dark blue, vfmacc in green). The reduction in vector instructions of the differently adapted versions compared to the stripped (original) one (which counted a total of 103,784 vector instructions) is indicated over the first bar. Numbers in the graph and table are expressed in thousands.

Our third and more complex method was using intrinsics, which, in some circumstances, can outperform the simple autovectorisation porting approach. The use of intrinsics allows the expansion of vectorisation, creating a pipeline of vector instructions in an outer loop, further reducing the number of memory accesses. This strategy reduces the vector instructions to 9,499, achieving an improvement of 13.4 × compared to the initial version.

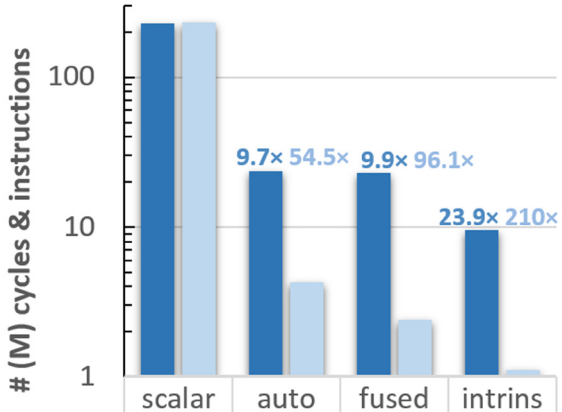
However, not all instructions have the same computational cost, so converting from the vector instructions counters to computing time (or speed up) may not be straightforward. We observe that our modifications managed to substantially reduce the number of memory accesses (loads (*vle*) and store (*vse*)), which are among the most computationally costly instructions. In fact, almost 2/3 of the *vle* and *vse* have been suppressed in the version with intrinsics. On the other hand, the accumulated number of fused multiply-add instructions (*vfmadd* and *vfmacc* are equivalent) remains almost constant. The settings of vector length (*vsetvli*) are also progressively reduced; however, their overall contribution to the total number of cycles is almost negligible compared to other instructions. In summary, we can conclude that our results show that all instruction types are being consistently reduced, guaranteeing the improved efficiency of the new implementation.

After our preliminary study with Vehave, we used an experimental platform to evaluate the RISC-V VPU, which is composed of an FPGA, and a host-x86 server used to program and communicate with it. In addition to measuring hardware counters such as the number of vector instructions, running on the FPGA allows one to obtain cycle-accurate time measurements. This metric enables a more straight-forward interpretation of how much the code's efficiency was improved at each implementation. The outcomes of our analyses are presented in Figure 2.

In this case, the scalar version exhibits almost a one-to-one ratio between cycles (227,443,896) and instructions (231,667,638). Compared to the scalar version, the autovectorised one reduces the cycles and instructions by a factor of 9.7 × and 54.5 ×, respectively. The efficiency was further improved in the version with fused loops, obtaining an overall speedup of 9.9 × in cycles and 96.1 × in instructions, and 23.9 × and 210 ×, respectively, using intrinsics.

Conclusions

In this short communication, we describe our efforts to inform the implementation of new (post)-exascale HPC systems based on RISC-V. Our porting is centred around the ELPA eigensolver, a library used by many of the most widely-used *ab initio* electronic structure codes. Our porting work was done on the RISC-V core with a VPU developed at the Barcelona



	scalar	auto	fused	intrins
cycles	227.444	23.475	22.866	9.499
instructions	231.668	4.254	2.411	1.103

Figure 2. Count of cycles (dark blue bars) and instructions (light blue) for the different vectorised versions of the ELPA mini-app executed on the FPGA system. The speed-up of the differently adapted versions compared to the scalar ones is indicated over the bars, following the same colour code. The y-axis is presented in a logarithmic scale, and all numbers are expressed in millions. PAPI counters were used for these implementations.

Supercomputing Center (BSC) within the framework of the European Processor Initiative (EPI)[30].

The manuscript summarises the iterative steps for improving the performance of a complex HPC library leveraging a RISC-V-based VPU prototype. Vectorisation of the kernel was achieved by (i) auto-vectorisation, (ii) fusing similar loops, and (iii) using intrinsics. This progressive approach offers several possibilities, from the most straightforward and portable approaches to the more complicated ones. The performance improvements are obtained with minimal intervention to the original code since we rely in the first instance on the autovectorisation features of the compiler. In addition to that, our vectorised code using intrinsics did not require us to implement loop unrollings, which would have increased the complexity of the code. Moreover, when relying on compiler autovectorisation, the code remains a plain C (or Fortran) code annotated with pragmas, so performance can be achieved without adding calls to external kernels, maintaining the code clean and portable. We should remark that one of the main particularities of this platform is that vector loads and stores are designed to bypass the (32-kByte) L1 cache. This architectural choice prevents large vector data, which can often exceed the cache's capacity, from polluting it content. Instead, this data is routed directly to the (larger, 1 MB) L2 cache. This design significantly reduces the complexity of managing the cache optimization, which is often one of the trickiest aspects of porting to new architectures. The platform's memory hierarchy supports faster and more predictable data access patterns, particularly in high-performance computations that involve large matrices or vectors. This characteristic allowed us to focus more on maximizing vectorization and parallelism, rather than needing to re-engineer data movement strategies between memory and cache. Our experience highlights how modern architectures, such as this one, can simplify the process of porting highly optimized kernels. Another important benefit of this architecture is the flexibility of the variable vector length, which allowed us to avoid the problem of dealing with the tails of the arrays, reducing the complexity of the codes. Our study implements standard vectorisation techniques that are widely applicable across various architectures. The RISC-V vector extension's flexibility, particularly its support for varying vector lengths, allowed us to explore long-vector implementations effectively. This characteristic is not unique to our platform, and similar capabilities are present in other architectures supporting long-vector processing, such as the NEC SX-Aurora TSUBASA. Moreover, the new v1.0 of our V-extension will allow the compilation of codes in both Fortran and C, so future porting of Fortran code will be more straightforward when the updated hardware is available.

We should note that the ELPA library has a patterned file that can be used to create specific kernels for new architectures. Therefore, while we will focus on a specific kernel, porting can be further replicated throughout the library following an analogous procedure. The experience gained provides practical guidance for other codes and architectures. So far, the performance of ELPA has been proven to be excellent for large matrices when GPUs are used, however, the CPU version has better performance for smaller problems. With the inclusion of vector architecture in a heterogeneous HPC system, the crossover point where the GPUs outperform could be potentially shifted toward larger matrix sizes. Ideally, one could imagine a homogeneous system made of EPAC-VEC where all performance improvement is delivered by the vector unit, making the development and maintance effort of the code simpler. In addition, our mini-app-based model represents a pragmatic, user-friendly approach to facilitate co-design efforts, cooperatively in fine-tuning the software and hardware components. The outcomes of these efforts will contribute significantly to advancing the porting of *ab initio* computational materials and molecular science codes - one of the most relevant families of applications with many users in the HPC community - to (post-)exascale hardware architectures developed in the EU. Furthermore, this evaluation serves as valuable feedback for hardware designers, system integrators and engineers actively involved in compilers for the systems.

Software availability

All the code developed for this work has been openly stored in Zenodo, https://doi.org/10.5281/zenodo.12665268[31], while the project is also openly available on our institutional BSC Materials Science group GitLab https://gitlab.bsc.es/material-science/elpa2riscv-public. The code is published under GNU LGPLv3 license: https://www.gnu.org/licenses/lgpl-3.0.html. The ELPA library is also open-source and can be obtained from https://elpa.mpcdf.mpg.de/. The specification of the RISC-V "V" extension v0.7.1 used in this work can be openly accessed at https://github.com/riscv/riscv-v-spec/releases/tag/0.7.1, under a Creative Commons Attribution 4.0 International Public License. The intrinsics used with the EPI compiler can also be accessed at https://admin.hca.bsc.es/epi/ftp/doc/intrinsics/EPI-0.7/epi-intrinsics.html.

Acknowledgements

We thank Andreas Marek and Petr Karpov, core developers of ELPA at the MPCDF, for their suggestions and for participating in fruitful discussions during the development of this project.

References

1. Oliveira MJT, Papior N, Pouillon Y, *et al.*: **The CECAM electronic structure library and the modular software development paradigm.** *J Chem Phys.* 2020; **153**(2): 024117.
PubMed Abstract | **Publisher Full Text**

2. Gavini V, Baroni S, Blum V, *et al.*: **Roadmap on electronic structure codes in the exascale era.** *Model Simul Mater Sci Eng.* 2023; **31**(6): 063301.
Publisher Full Text

3. Marek A, Blum V, Johanni R, *et al.*: **The ELPA library: scalable parallel**

eigenvalue solutions for electronic structure theory and computational science. *J Phys Condens Matter.* 2014; **26**(21): 213201.
PubMed Abstract | Publisher Full Text

4. Kůs P, Marek A, Köcher SS, *et al.*: **Optimizations of the eigensolvers in the ELPA library.** *Parallel Comput.* 2019; **85**: 167–177.
Publisher Full Text

5. Yu VWZ, Moussa J, Kůs P, *et al.*: **GPU-acceleration of the ELPA2 distributed eigensolver for dense symmetric and hermitian eigenproblems.** *Comput Phys Commun.* 2021; **262**: 107808.
Publisher Full Text

6. Kresse G, Furthmüller J: **Efficiency of *ab-initio* total energy calculations for metals and semiconductors using a plane-wave basis set.** *Comp Mater Sci.* 1996; **6**(1): 15–50.
Publisher Full Text

7. García A, Papior N, Akhtar A, *et al.*: **Siesta: recent developments and applications.** *J Chem Phys.* 2020; **152**(20): 204108.
PubMed Abstract | Publisher Full Text

8. Giannozzi P, Andreussi O, Brumme T, *et al.*: **Advanced capabilities for materials modelling with Quantum ESPRESSO.** *J Phys Condens Matter.* 2017; **29**(46): 465901.
PubMed Abstract | Publisher Full Text

9. Gonze X, Jollet F, Abreu Araujo F, *et al.*: **Recent developments in the ABINIT software package.** *Comput Phys Commun.* 2016; **205**: 106–131.
Publisher Full Text

10. Gulans A, Kontur S, Meisenbichler C, *et al.*: **Exciting: a full-potential all-electron package implementing Density-Functional Theory and many-body perturbation theory.** *J Phys Condens Matter.* 2014; **26**(36): 363202.
PubMed Abstract | Publisher Full Text

11. Blum V, Gehrke R, Hanke F, *et al.*: ***Ab initio* molecular simulations with numeric atom-centered orbitals.** *Comput Phys Commun.* 2009; **180**(11): 2175–2196.
Publisher Full Text

12. Mortensen JJ, Larsen AH, Kuisma M, *et al.*: **GPAW: an open Python package for electronic structure calculations.** *J Chem Phys.* 2024; **160**(9): 092503.
PubMed Abstract | Publisher Full Text

13. Kühne TD, Iannuzzi M, Del Ben M, *et al.*: **CP2K: an electronic structure and molecular dynamics software package - Quickstep: efficient and accurate electronic structure calculations.** *J Chem Phys.* 2020; **152**(19): 194103.
PubMed Abstract | Publisher Full Text

14. Yu VWZ, Campos C, Dawson W, *et al.*: **ELSI — an open infrastructure for electronic structure solvers.** *Comput Phys Commun.* 2020; **256**: 107459.
Publisher Full Text

15. Zhang L, Kozhevnikov A, Schulthess T, *et al.*: **Performance enhancement of APW+*lo* calculations by simplest separation of concerns.** *Computation.* 2022; **10**(3): 43.
Publisher Full Text

16. Gutiérrez Moreno JJ: ***Ab initio* guided atomistic modelling of nanomaterials on exascale High-Performance Computing platforms.** *Nano Futures.* 2024; **8**(1): 012501.
Publisher Full Text

17. Utz-Uwe Haus HPE, Laure E, Narasimhamurthy S, *et al.*: **Heterogeneous High Performance Computing.** ETP4HPC White Paper. 2022.
Reference Source

18. Minervini F, Palomar O, Unsal O, *et al.*: **Vitruvius+: an area-efficient RISC-V decoupled vector coprocessor for High Performance Computing applications.** *ACM Trans Archit Code Optim.* 2023; **20**(2): 28.
Publisher Full Text

19. Vizcaino P, Mantovani F, Ferrer R, *et al.*: **Acceleration with long vector architectures: implementation and evaluation of the FFT kernel on NEC SX-Aurora and RISC-V vector extension.** *Concurrency Computat Pract Exper.* 2023; **35**(20): e7424.
Publisher Full Text

20. Gómez C, Mantovani F, Focht E, *et al.*: **Efficiently running SpMV on long vector architectures.** In: *Proceedings of the 26th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming.* 2021; 292–303.
Publisher Full Text

21. Vizcaino P, Labarta J, Mantovani F: **Graph computing on long vector architectures (yes, it works!).** In: *2024 IEEE International Parallel and Distributed Processing Symposium (IPDPS).* IEEE, Accepted, in press, 2024.
Publisher Full Text

22. Blancafort M, Ferrer R, Houzeaux G, *et al.*: **Exploiting long vectors with a CFD code: a co-design show case.** In: *2024 IEEE International Parallel and Distributed Processing Symposium (IPDPS).* IEEE, Accepted, in press, 2024.
Publisher Full Text

23. Mantovani F, Vizcaino P, Banchelli F, *et al.*: **Software Development Vehicles to enable extended and early co-design: a RISC-V and HPC case of study.** In: Amanda Bienz, Michèle Weiland, Marc Baboulin, and Carola Kruse, editors, *High Performance Computing.* Cham, Springer Nature Switzerland, 2023; 526–537.
Publisher Full Text

24. **EPI project - LLVM-based compiler.** Accessed: 2024-06-21.
Reference Source

25. Pillet V, Labarta J, Cortes T, *et al.*: **PARAVER: a tool to visualize and analyze parallel code.** *WoTUG-18.* 1995; **44**: 03.
Reference Source

26. **EPI intrinsics.** Accessed: 2024-07-19.
Reference Source

27. **NOMAD center of excellence.** Accessed: 2024-07-19.
Reference Source

28. Mas Magre I, Torres Grima R, Cela Espin JM, *et al.*: **The NOMAD mini-apps: a suite of kernels from ab initio electronic structure codes enabling co-design in High-Performance Computing [version 3; peer review: 3 approved].** *Open Res Eur.* 2025; **4**: 35.
PubMed Abstract | Publisher Full Text | Free Full Text

29. Vizcaino P, Ieronymakis G, Dimou N, *et al.*: **Short reasons for long vectors in HPC CPUs a study based on RISC-V.** In: *Proceedings of the SC '23 Workshops of The International Conference on High Performance Computing, Network, Storage, and Analysis*. 2023; 1543–1549.
Publisher Full Text

30. **EPI - The European Processor Initiative.** Accessed: 2024-07-19.
Reference Source

31. Grima R, Vizcaíno P, Mantovani F, *et al.*: **ELPA2RISC-V.** *Zenodo*. [Code], 2024.
http://www.doi.org/10.5281/zenodo.12665269

Open Peer Review

Current Peer Review Status: ✔ ✔ ✔ ✔ ✔

Version 4

Reviewer Report 23 September 2025

https://doi.org/10.21956/openreseurope.21575.r59769

© **2025 Takahashi K.** This is an open access peer review report distributed under the terms of the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

✔ **Keichi Takahashi**
D3 Center, University of Osaka, Osaka, Japan

The authors have addressed all of my concerns and questions in their revised manuscript and response. I especially appreciated the additional analysis of optimality using hardware counters.

Competing Interests: No competing interests were disclosed.

Reviewer Expertise: HPC, vector processors, GPU

I confirm that I have read this submission and believe that I have an appropriate level of expertise to confirm that it is of an acceptable scientific standard.

Version 3

Reviewer Report 10 December 2024

https://doi.org/10.21956/openreseurope.20502.r47178

© **2024 Takahashi K.** This is an open access peer review report distributed under the terms of the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

? **Keichi Takahashi**
D3 Center, University of Osaka, Osaka, Japan

This paper presents the porting effort of an eigensolver kernel to a RISC-V processor prototype targeting post-exascale computing. The kernel is extracted from the ELPA eigensolver library, which is used by various *ab initio* electron structure codes. The target processor, EPAC-VEC,

features a vector processing unit with 16,384-bit wide vector registers. The authors employ a progressive approach for vectorization: (1) auto-vectorization, (2) loop fusion, and (3) the use of intrinsics, where performance is improved at the cost of lower portability and higher implementation effort. The final version resulted in 13.4x fewer vector instructions and 23.9x fewer clock cycles.

- What is the optimality of the mini-app after porting? The paper demonstrates improvements over the initial version of the mini-app, but does not show whether the final version fully exploits the potential of the EPAC-VEC processor. I suggest adding an assessment of the optimality, such as a roofline analysis and/or vectorization rate.
- Although the paper is entitled "Co-designing", it assumes the hardware architecture as given and only describes a unidirectional effort: adapting a software to the target platform. I suggest clarifying what insights or recommendations for the hardware designers arise based on the results of this paper.
- The employed progressive approach is a widely adopted when vectorizing applications. I suggest highlighting the porting techniques that are specific to the RISC-V vector extension or EPAC-VEC, if there are any.
- I could not follow the discussion on cache in the conclusion: "...one of the main particularities of this platform is that vector loads and stores do not pollute the cache." Does this mean all vector loads/stores bypass the cache? I suggest describing the cache design of EPAC-VEC earlier, or removing this discussion from the conclusion.

Is the work clearly and accurately presented and does it cite the current literature?
Yes

Is the study design appropriate and does the work have academic merit?
Yes

Are sufficient details of methods and analysis provided to allow replication by others?
Yes

If applicable, is the statistical analysis and its interpretation appropriate?
Yes

Are all the source data underlying the results available to ensure full reproducibility?
Yes

Are the conclusions drawn adequately supported by the results?
Yes

Competing Interests: No competing interests were disclosed.

Reviewer Expertise: HPC, vector processors, GPU

I confirm that I have read this submission and believe that I have an appropriate level of expertise to confirm that it is of an acceptable scientific standard, however I have significant reservations, as outlined above.

Author Response 04 Sep 2025

julio gutierrez

We are very grateful for the referee's careful reading and constructive suggestions. Although we received this last review report once the paper had already been formally accepted, we believe the additional comments offer valuable insights that can strengthen the final version of the manuscript. In the spirit of scientific discussion, we have addressed his feedback by updating the conclusions. We hope these revisions further enhance the quality and clarity of the paper.

Comment#1: What is the optimality of the mini-app after porting? The paper demonstrates improvements over the initial version of the mini-app, but does not show whether the final version fully exploits the potential of the EPAC-VEC processor. I suggest adding an assessment of the optimality, such as a roofline analysis and/or vectorization rate.
Response: We thank the reviewer for this insightful suggestion. We fully agree that assessing the degree of optimality is important to evaluate how effectively the application exploits the architecture. However, our current work is based on an FPGA implementation of the EPAC-VEC processor, where performance metrics such as peak frequency or memory bandwidth are not yet stable or representative of a finalized platform. This makes traditional roofline models or absolute performance comparisons less meaningful at this stage.
That said, we have conducted a preliminary analysis based on hardware counters. The mini-app achieved full vectorisation, with no tail effects, and the measured cycles per vector instruction (~36.4) closely match the expected latency for vector fused multiply-add operations (~35 cycles for 256 double-precision elements), which dominate the workload. This suggests the application is already utilizing the vector unit efficiently and approaching the latency-limited performance of the current implementation.
A more complete evaluation of performance relative to architectural ceilings, including roofline analysis, will be more appropriate once the architecture matures into an ASIC form with well-defined performance bounds. We view this as an important direction for future work.

Comment#2: Although the paper is entitled "Co-designing", it assumes the hardware architecture as given and only describes a unidirectional effort: adapting a software to the target platform. I suggest clarifying what insights or recommendations for the hardware designers arise based on the results of this paper.
Response: The performance metrics obtained served as empirical data that can inform both compiler optimisations and hardware design decisions. This collaboration led to enhancements in the compiler's ability to optimise code for the RISC-V vector architecture, thereby improving overall system performance. Such iterative refinement between software requirements and compiler capabilities exemplifies the co-design paradigm, where insights from software adaptation inform and enhance compiler and hardware design. This feedback loop is essential in co-design, enabling hardware architects and compiler developers to tailor future architectures and tools to better accommodate the computational patterns observed in scientific applications.

Comment#3: The employed progressive approach is a widely adopted when vectorizing applications. I suggest highlighting the porting techniques that are specific to the RISC-V vector extension or EPAC-VEC, if there are any.
Response: Our study implements standard vectorisation techniques that are widely applicable across various architectures. The RISC-V vector extension's flexibility, particularly its support for varying vector lengths, allowed us to explore long-vector implementations effectively. The RISC-V vector extension's flexible design facilitates efficient vectorisation, particularly due to its support for varying vector lengths, enabling the exploration of long-vector implementations without necessitating architecture-specific techniques. This characteristic is not unique to RISC-V, and similar capabilities are present in other architectures, such as the NEC SX-Aurora TSUBASA. Therefore, the porting techniques we employed are not exclusive to RISC-V but are applicable to any architecture that supports long-vector processing.

Comment#4: I could not follow the discussion on cache in the conclusion: "...one of the main particularities of this platform is that vector loads and stores do not pollute the cache." Does this mean all vector loads/stores bypass the cache? I suggest describing the cache design of EPAC-VEC earlier, or removing this discussion from the conclusion.
Response: Vector memory accesses bypass L1 cache by default. L1 cache size is 32 kBytes: Caching the data of a few vector instructions would wipe the whole content of the cache, so it is not convenient for the scalar core. L2 cache size is 1 MB, and it is also used as a cache for data involved in vector instructions. The conclusions section has been updated to include these last two clarifications.

Competing Interests: No competing interests were disclosed.

Version 2

Reviewer Report 05 November 2024

https://doi.org/10.21956/openreseurope.20353.r45574

© **2024 Dawson W.** This is an open access peer review report distributed under the terms of the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

William Dawson
RIKEN Center for Computational Science, Kobe, Japan

The response and updated manuscript thoroughly covers the concerns I had with the initial draft. I especially think that the new conclusion and various added pieces addressed my main issue about a lack of broad conclusions. Therefore I recommend this for indexing.
Regarding the license: now the text still says GPLv3, and the repository has the GPLv3 license. But my impression was that both would ideally be LGPLv3, since that is the license of ELPA.

If the authors decide to make further revisions, I have a few additional small comments about the style:
* Page 3: "Density Functional theory" - theory should be capitalized
* Page 3: cubic scalability -> cubic scaling growth in computational costs
* Page 4: break up the long paragraph into several smaller ones
* Page 4: "serve both to different purposes" -> "serve different purposes"
* Page 6: "with a minimal intervention" -> "with minimal intervention"
* Page 6: "did not require to" -> "did not require us to"
* Page 6: "the CPU version outperforms" -> "the CPU version has better performance"
* Page 6: "more users in the HPC" -> "many users in the HPC"
* The link to "https://gitlab.bsc.es/material-science/elpa2riscv-public" is broken, containing an extra space before material

Competing Interests: I confirm that this potential conflict of interest did not affect my ability to write an objective and unbiased review of the article.

Reviewer Expertise: Electronic structure calculations, high performance computing.

I confirm that I have read this submission and believe that I have an appropriate level of expertise to confirm that it is of an acceptable scientific standard.

Author Response 12 Nov 2024

julio gutierrez

We much appreciate the comments provided again by the referee. Although the license was updated to LGPLv3 in the previous revision, this change was only visible in the repository and was not correctly indicated in the manuscript. We have fixed this and the link to our GitLab. We have also implemented the additional wording suggestions.

Competing Interests: No competing interests were disclosed.

Version 1

Reviewer Report 04 September 2024

https://doi.org/10.21956/openreseurope.19799.r43456

© 2024 Dawson W. This is an open access peer review report distributed under the terms of the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

?

William Dawson

RIKEN Center for Computational Science, Kobe, Japan

In this paper, Torres et al. describe some early experience porting the ELPA eigenvalue solver library to a RISC-V platform featuring a wide vector unit (as part of the European Processor Initiative). They analyze how efficient the compiler generated code is, and how it can be improved through a sequence of tuning steps.

I appreciated the opportunity to read the paper. This is a valuable article for the community, as they are targeting what may be a very relevant platform in the future. The article is cross disciplinary, yet very approachable and the main points are easy to understand. Given its forward-looking nature and narrow scope, it seems appropriate for a brief report.

The main weakness of this paper is that the authors don't provide any broad, transferable results from this data. If I was tasked with the porting ELPA to a production ready RISC-V supercomputer (or to any new machine), I would do exactly as the authors did: I would consider what compiler flags might be optimal, I would manually fuse loops, I would write intrinsic vectorization instructions, etc. Thus, this paper only describes the basic state-of-the-art, instead of going one step further and really evaluating the platform and creating new knowledge.

I recommend that the authors rewrite the conclusion section, which wastes half a page repeating information that is fresh in the mind of the reader of this short report. Instead, I would like the authors to make broad recommendations from their data such as: how should developers prepare their code for the prospect of being ported to this platform, what weaknesses exist in the current compiler toolchains for this system, what other codes should be evaluated based on the weaknesses found here, should we expect this platform to perform well in comparison to other platforms for ELPA if optimization is done properly, if a supercomputer is built on this platforms and runs realistic workflows will this kernel obtain peak performance in this kernel, etc. These are just suggestions, not a to do list; I think the authors will know best which broad conclusions are best justified by the data.

As you note, the authors of ELPA made extensive efforts to specialize for previous platforms, so what's new about the porting experience for this platform? (or maybe nothing is, and your conclusion is that this platform works great if you use standard techniques). This kind of analysis will really allow the authors to impact the community by leveraging their experience and expertise.

Two other broad criticisms I have are:

1. It is not clear to me why the authors use both the Vehave emulator and the FPGA implementation. What are the strengths of the emulator in this regard? Why not just use the (cycle accurate) FPGA? Are the systems tested the same in both cases? Maybe a broad takeaway from this paper could be about which tools to use.
2. The conversion of the code from Fortran to C risks making this benchmark less realistic. Maybe when the Fortran compiler for this platform becomes available, the compiler would be able to do these optimizations without any problem. If you try to compile the C version you wrote on other platforms, how well does it perform against the Fortran version? Maybe your C conversion introduced artificial challenges for the compiler. I think some checks about this would make the paper more solid.

In addition to these broad recommendations, I have some other small comments which the

authors might consider for improving their paper.

- In the abstract, just calling Vehave “the user-space Vehave” is unclear, you should write that it’s an emulator there.
- In the introduction, they cite a roadmap article. Since the introduction specifically talks about modular design, they may also consider citing an article describing the CECAM effort [1]
- When describing the importance of eigenvalue problems, the authors should clarify that ELPA targets the dense eigenvalue problem, to distinguish it from sparse solvers (though ELPA is used as a building block for sparse solvers).
- The authors specifically say that eigenvalue solvers take “90% of the total computational time in relatively large systems”. Since they have a specific number there, they should cite where they got it from.
- The authors should describe more specifically what they are testing on. “ELPA checkpoints for matrices with different sizes are accessible” – but what size are you using here? Do you use all the data or just one specific problem?
- I am pretty sure the data you have labeled “auto” is after the Q-stripes are suppressed, but if you could explicitly note this point, I think that’d be clearer.
- For Fig. 2, it could help to repeat the color scheme of Fig. 1 instead of using two shades of blue. I also liked that in Fig. 1 you had a patch with the colors in the table, you should do that in Fig. 2 as well.
- In the paper you claim the code is published a GPL, but on the repository it’s LGPL (as it should be so that code can be backported to ELPA).
- Reference 24 mistakenly lists “Departament Computadors” as an author.

Some grammatical / spelling / stylistic corrections I suggest:

- Fig. 2 table: clycles => “cycles”.
- Various places: vectorial instructions => “vectorized instructions” or sometimes “vector instructions” depending on the context, vectorial compiler => vectorizing compiler, etc.
- Introduction, paragraph 1: easily adapted to incorporate => more adaptable to the incorporation of
- Introduction, paragraph 1: and to perform efficiently heterogeneous computing systems. => and to perform efficiently “on” heterogeneous computing systems.
- Introduction, paragraph 2: FHI-aims, GPAW, or CP2K => FHI-aims, GPAW, “and” CP2K.
- Introduction, paragraph 2: The library can be directly implemented => The library can be directly “integrated”
- Introduction, paragraph 3: a significant advancement HPC => a significance advancement “in” HPC
- Introduction, paragraph 4: the relevance of the eigensolver => the relevance of eigensolvers
- Introduction, paragraph 4: landmark work => pioneering works
- Methodology, paragraph 2: runs on top of the RISC-V => runs on top of RISC-V
- Methodology, paragraph 2: This FPGA is used as user-defined => This FPGA is used as a user-defined.
- Methodology paragraph 2: this link => this looks weird with the superscript citations, maybe replace “this link” with “online”.
- Methodology paragraph 3: computationally cost => computational cost.
- Methodology paragraph 3: especially to the extensive => especially for the extensive
- Methodology paragraph 3: the indications to execute => the instructions to execute

- Methodology paragraph 3: associated online repositories. Here a link would be good (I know it comes at the end of the article, but it'd be convenient to have it here too).
- Results and discussion paragraph 3: The increasing performance. I don't understand what this sentence means. I also don't understand why you characterize your strategy as bottom-up.
- Results and discussion paragraph 4: counted for a total => counted a total.
- Results and discussion paragraph 4: without limitations of 48 elements => without the limitation of 48 elements
- Results and discussion paragraph 6: either put a comma or a colon after 9,499.
- Results and discussion paragraph 7: On the other hand, => I was confused by this part until I realized you meant the sum of vfmadd and vfmacc remains constant, not each one individually.
- Results and discussion paragraph 8: RISC-V VPU, composed of an FPGA, and a => RISC-V VPU, which is composed of an FPGA and a

I don't have access to a RISC-V machine, but I tested out your source code in the gitlab repository on an x86 machine. I think the instructions were easy to follow, but I had some questions:

- M_REAL_SIMPLE_F – what optimizations are done here? It's faster than M_REAL_REFERENCE, but slower than M_REAL_SIMPLE.
- What is the difference between GENERIC and SIMPLE? Why is GENERIC slower than simple?

References

1. Oliveira MJT, Papior N, Pouillon Y, Blum V, et al.: The CECAM electronic structure library and the modular software development paradigm.*J Chem Phys*. 2020; **153** (2): 024117 PubMed Abstract | Publisher Full Text

Is the work clearly and accurately presented and does it cite the current literature?

Yes

Is the study design appropriate and does the work have academic merit?

Yes

Are sufficient details of methods and analysis provided to allow replication by others?

Yes

If applicable, is the statistical analysis and its interpretation appropriate?

Yes

Are all the source data underlying the results available to ensure full reproducibility?

Yes

Are the conclusions drawn adequately supported by the results?

Yes

Competing Interests: I confirm that this potential conflict of interest did not affect my ability to write an objective and unbiased review of the article.

Reviewer Expertise: Electronic structure calculations, high performance computing.

I confirm that I have read this submission and believe that I have an appropriate level of expertise to confirm that it is of an acceptable scientific standard, however I have significant reservations, as outlined above.

Author Response 23 Oct 2024

julio gutierrez

Comment #1: In this paper, Torres et al. describe some early experience porting the ELPA eigenvalue solver library to a RISC-V platform featuring a wide vector unit (as part of the European Processor Initiative). They analyze how efficient the compiler generated code is, and how it can be improved through a sequence of tuning steps. I appreciated the opportunity to read the paper. This is a valuable article for the community, as they are targeting what may be a very relevant platform in the future. The article is cross disciplinary, yet very approachable and the main points are easy to understand. Given its forward-looking nature and narrow scope, it seems appropriate for a brief report. The main weakness of this paper is that the authors don't provide any broad, transferable results from this data. If I was tasked with the porting ELPA to a production ready RISC-V supercomputer (or to any new machine), I would do exactly as the authors did: I would consider what compiler flags might be optimal, I would manually fuse loops, I would write intrinsic vectorization instructions, etc. Thus, this paper only describes the basic state-of-the-art, instead of going one step further and really evaluating the platform and creating new knowledge.

Answer: We would like to referee for his thorough revision and the suggested comments and corrections, that we believe have help us to improve the manuscript substantially. The revision was made according to the referee's requirements and we answered all points made. Individual responses are given after each comment. Before addressing the following points, we would like to clarify that the main novelty of our work lays on the use of this novel RISC-V VPU (and its associated compiler suite) to solve the eigenvalue problem. One of the main particularities of this platform is that vector loads and stores do not pollute the cache. This significantly reduces the complexity of managing the cache optimization, which is often one of the trickiest aspects of porting to new architectures. The platform's memory hierarchy supports faster and more predictable data access patterns, particularly in high-performance computations that involve large matrices or vectors. This characteristic allowed us to focus more on maximizing vectorization and parallelism, rather than needing to re-engineer data movement strategies between memory and cache. Our experience highlights how modern architectures, such as this one, can simplify the process of porting highly optimized kernels. The lessons learned—particularly the ease of porting due to efficient vector register handling—will help inform future efforts in porting scientific computing libraries to similar platforms. The reduced complexity around cache management is a particularly valuable insight that we believe will be beneficial to the broader HPC community.

Comment #2: I recommend that the authors rewrite the conclusion section, which wastes half a page repeating information that is fresh in the mind of the reader of this short

report. Instead, I would like the authors to make broad recommendations from their data such as: how should developers prepare their code for the prospect of being ported to this platform, what weaknesses exist in the current compiler toolchains for this system, what other codes should be evaluated based on the weaknesses found here, should we expect this platform to perform well in comparison to other platforms for ELPA if optimization is done properly, if a supercomputer is built on this platforms and runs realistic workflows will this kernel obtain peak performance in this kernel, etc. These are just suggestions, not a to do list; I think the authors will know best which broad conclusions are best justified by the data.

Answer: There are two main advantages in following the approach shared in this paper:
1. The performance improvements are obtained with a minimal intervention to the original code since we rely in the first instance on the autovectorisation features of the compiler. In addition to that, our vectorised code using intrinsics did not require to implement loop unrollings, which would have increased the complexity of the code.
2. When relying on compiler autovectorization, the code remains a plain C (or Fortran) code annotated with pragmas. This means that performance can be achieved without adding calls to external kernels (e.g., no CUDA dependencies) thus maintaining the code clean and portable. Another important benefit of this architecture is the flexibility of the variable vector length, which allowed us to avoid the problem of dealing with the tails of the arrays, reducing the complexity of the codes.
So far, the performance of ELPA has been proven to be excellent for large matrices when GPUs are used, however, the CPU version outperforms for smaller problems. With the inclusion of vector architecture in a heterogeneous HPC system, the crossover point where the GPUs outperform could be potentially shifted toward larger matrix sizes. Ideally, one could imagine a homogeneous system made of EPAC-VEC where all performance improvement is delivered by the vector unit, making the development and maintance effort of the code simpler.
We have tested EPAC-VEC for other scientific software applications such as Alya, a fluid dynamic code developed in Fortran and we demonstrated that a significant speed-up (up to 7×) can be achieved with code modifications that leave the code portable and deliver performance improvements also on Intel x86 architectures (while leveraging AVX512 SIMD extension by Intel). [Blancafort, Marc, et al. "Exploiting long vectors with a CFD code: a co-design show case." 2024 IEEE International Parallel and Distributed Processing Symposium (IPDPS)]. The EPI team also proved that the EPAC-VEC architecture enables significant performance improvement when dealing with non-dense linear algebra workloads. In this paper, for example, Sparse-matrix algebra and kernels are considered [Vizcaino, Pablo, et al. "Short reasons for long vectors in HPC CPUs: a study based on RISC-V." Proceedings of the SC'23 Workshops of The International Conference on High Performance Computing, Network, Storage, and Analysis. 2023. https://arxiv.org/abs/2309.06865]. However, for workloads such as graph analytics or FFT computation a previous rework of the algorithm for accessing the data is required. Still, this work can be encapsulated into “vectorised libraries” and made available to the user, making the process of porting smoother. A key role in the process of testing and adopting the EPAC-VEC architecture is played by the tools used for analysing the vectorization achieved. The mental scheme of the scientist can create conceptual barriers when analysing the code. For this reasons a set of tools have been developed for analysing vectorization of the code and understand potential for

vectorization [Vizcaino, Pablo, et al. "RAVE: RISC-V Analyzer of Vector Executions, a QEMU tracing plugin." arXiv preprint arXiv:2409.13639 (2024). https://arxiv.org/abs/2409.13639]
The manuscript has been revised in several key areas.
The summary part included in the conclusions section has been shortened. Additionally, various comments regarding outlook aspects and specific details about the platform have been incorporated into the text. Furthermore, the description of the Alya use case has been expanded in the introduction.

Comment #3: As you note, the authors of ELPA made extensive efforts to specialize for previous platforms, so what's new about the porting experience for this platform? (or maybe nothing is, and your conclusion is that this platform works great if you use standard techniques). This kind of analysis will really allow the authors to impact the community by leveraging their experience and expertise.

Answer: Each new platform presents its own set of challenges and opportunities, particularly when it comes to fully leveraging the architecture's unique characteristics. As it was mentioned before in the response, one of the most unique features of this platform is that vector loads and stores do not pollute the cache, which simplifies managing the cache optimizations, being these particularly critical in computations involving large matrices and/or vectors. This characteristic would allow the developer to focus more on maximizing vectorization and parallelism, rather than re-engineer data movement strategies between memory and cache. This has been highlighted in the conclusions section.

Comment #4: Two other broad criticisms I have are:

1. ***It is not clear to me why the authors use both the Vehave emulator and the FPGA implementation. What are the strengths of the emulator in this regard? Why not just use the (cycle accurate) FPGA? Are the systems tested the same in both cases? Maybe a broad takeaway from this paper could be about which tools to use.***
2. ***The conversion of the code from Fortran to C risks making this benchmark less realistic. Maybe when the Fortran compiler for this platform becomes available, the compiler would be able to do these optimizations without any problem. If you try to compile the C version you wrote on other platforms, how well does it perform against the Fortran version? Maybe your C conversion introduced artificial challenges for the compiler. I think some checks about this would make the paper more solid.***

Answers:
1. The Vehave emulator and the FPGA implementation serve both to different purposes and offers unique advantages. The Vehave emulator is primarily used for flexibility during the early stages of development. It allows for quick iteration, debugging, and testing of different configurations without needing to reconfigure hardware. This is particularly useful when exploring architectural optimisations or validating functionalities under varying conditions. The emulator also provides a high-level view of performance trends and functionality before transitioning to hardware-based testing. On the other hand, the FPGA implementation offers cycle-accurate performance and power consumption data. While emulators are useful for early testing and development, FPGAs enable us to test the design in real-world conditions with accurate timing, ensuring that the system meets performance

targets in a hardware environment. Furthermore, the FPGA validates that our design operates as expected when mapped onto a physical implementation. This clarification has been added to the methods section.
2. Regarding the porting from Fortran to C, and connecting also with 4.1, we should clarify that at the time this work was carried out, we could work on Vehave using the latest v1.0 implementation of EPI, while the FPGA was on the previous v0.7.1. This new v1.0 of the V-extension allowed the compilation of codes in both Fortran and C. While the code porting in this case was extremely simple and straightforward, we also used the Vehave on v1.0 to extract traces and verify that both codes were equivalent. Whereas, of course, the binaries produced from the two codes where different, the number and type of vector instructions produced by both versions were proven to be analogous. This point has been clarified in the text.
Also, it is important to note that the original distribution of ELPA already contains two types of kernels: generic kernels written in Fortran and optimized kernels written in C. The C kernels are highly optimized for specific architectures and are used to achieve maximum performance on those platforms. The Fortran kernels, on the other hand, are generic and designed to be portable across different systems. Our implementation is based on a generic Fortran kernel, however, it is important to clarify these ones are not used by default in modern machines. On current architectures, the default kernels are specifically optimized for the hardware and are written in C. These optimized C kernels are designed to leverage platform-specific features for improved performance, which aligns with the approach we followed in our work.

Comment #5: In addition to these broad recommendations, I have some other small comments which the authors might consider for improving their paper.

- ***In the abstract, just calling Vehave "the user-space Vehave" is unclear, you should write that it's an emulator there.***
- ***In the introduction, they cite a roadmap article. Since the introduction specifically talks about modular design, they may also consider citing an article describing the CECAM effort [Oliveira MJT, Papior N, Pouillon Y, et al.: The CECAM electronic structure library and the modular software development paradigm. J Chem Phys. 2020; 153 (2): 024117]***
- ***When describing the importance of eigenvalue problems, the authors should clarify that ELPA targets the dense eigenvalue problem, to distinguish it from sparse solvers (though ELPA is used as a building block for sparse solvers).***
- ***The authors specifically say that eigenvalue solvers take "90% of the total computational time in relatively large systems". Since they have a specific number there, they should cite where they got it from.***
- ***The authors should describe more specifically what they are testing on. "ELPA checkpoints for matrices with different sizes are accessible" – but what size are you using here? Do you use all the data or just one specific problem?***
- ***I am pretty sure the data you have labeled "auto" is after the Q-stripes are suppressed, but if you could explicitly note this point, I think that'd be clearer.***
- ***For Fig. 2, it could help to repeat the color scheme of Fig. 1 instead of using two shades of blue. I also liked that in Fig. 1 you had a patch with the colors in the table, you should do that in Fig. 2 as well.***
- ***In the paper you claim the code is published a GPL, but on the repository it's LGPL (as***

it should be so that code can be backported to ELPA).

- ***Reference 24 mistakenly lists "Departament Computadors" as an author.***

Answer: We thank the reviewer for all these corrections and suggestions of improvements. All the indicated corrections have been implemented in the text. In addition, we would like to clarify that:

- The estimation of the relative computational time spent in the eigensolver is mostly based on our extensive user experience, going across several electronic structure codes over more than a decade. This has been clarified in the text.
- ELPA is shipped along with a variety of mini-apps to test the different kernels and precision levels on several types of matrices. We were interested in the one that gets the eigenvectors of a real double-precision matrix using the 2-stage method solver for a random matrix. We saved different checkpoints of the application at the entrance of the function *trans_ev_tridi_to_band*. Each checkpoint is labeled with the matrix size, the block size, and the number of eigenvectors that are computed. Note that we have vectorised our code in the direction of the number of eigenvectors and it is very sensitive to this value.

For the sake of correctness, we tested all the kernels of our code using Vehave, using every available checkpoint. To test the vector behavior, we focused on our most relevant cases, with large matrix sizes and a number of eigenvectors greater than the vector length. The default checkpoint file is a matrix of 1024 elements, 256 eigenvectors and a blocksize of 32 elements. This has been clarified at the end of the methods section.

- We agree that not mentioning that our reference version does not have Q-stripes could be misleading. This information has been clarified in Fig.1's caption.
- Figure 1, the blue bars are used represented the total number of instructions for each code version, while other colours are used for partial contributions. Following that same reasoning, blue is used for the bars in Figure 2, which show also total number of cycles and instructions. We believe that using an identical colour scheme, although more visual, could also confuse the reader. In any case, the colours have been updated with brighter tones, the patches in the table have also been added to Figure 2, and the typo identified in the next block of comments has been corrected.
- Our repositories have been updated to LGPL-v3.

Comment #6: Some grammatical / spelling / stylistic corrections I suggest:

- ***Fig. 2 table: clycles => "cycles".***
- ***Various places: vectorial instructions => "vectorized instructions" or sometimes "vector instructions" depending on the context, vectorial compiler => vectorizing compiler, etc.***
- ***Introduction, paragraph 1: easily adapted to incorporate => more adaptable to the incorporation of***
- ***Introduction, paragraph 1: and to perform efficiently heterogeneous computing systems. => and to perform efficiently "on" heterogeneous computing systems.***
- ***Introduction, paragraph 2: FHI-aims, GPAW, or CP2K => FHI-aims, GPAW, "and" CP2K.***
- ***Introduction, paragraph 2: The library can be directly implemented => The library can be directly "integrated"***
- ***Introduction, paragraph 3: a significant advancement HPC => a significance advancement "in" HPC***
- ***Introduction, paragraph 4: the relevance of the eigensolver => the relevance of***

eigensolvers

- ***Introduction, paragraph 4: landmark work => pioneering works***
- ***Methodology, paragraph 2: runs on top of the RISC-V => runs on top of RISC-V***
- ***Methodology, paragraph 2: This FPGA is used as user-defined => This FPGA is used as a user-defined.***
- ***Methodology paragraph 2: this link => this looks weird with the superscript citations, maybe replace "this link" with "online".***
- ***Methodology paragraph 3: computationally cost => computational cost.***
- ***Methodology paragraph 3: especially to the extensive => especially for the extensive***
- ***Methodology paragraph 3: the indications to execute => the instructions to execute***
- ***Methodology paragraph 3: associated online repositories. Here a link would be good (I know it comes at the end of the article, but it'd be convenient to have it here too).***
- ***Results and discussion paragraph 3: The increasing performance. I don't understand what this sentence means. I also don't understand why you characterize your strategy as bottom-up.***
- ***Results and discussion paragraph 4: counted for a total => counted a total.***
- ***Results and discussion paragraph 4: without limitations of 48 elements => without the limitation of 48 elements***
- ***Results and discussion paragraph 6: either put a comma or a colon after 9,499.***
- ***Results and discussion paragraph 7: On the other hand, => I was confused by this part until I realized you meant the sum of vfmadd and vfmacc remains constant, not each one individually.***
- ***Results and discussion paragraph 8: RISC-V VPU, composed of an FPGA, and a => RISC-V VPU, which is composed of an FPGA and a***

Answer: We thank the reviewer for all these corrections and suggestions of improvements. The suggested corrections have been incorporated to the text.

- The link to the mini-apps repository has been added.
- We would like to clarify that we characterized our strategy as bottom-up because it starts with the least effort and gradually increases in complexity. This approach is practical because it allows for incremental improvements and ensures that simpler, more portable solutions are considered before resorting to more complex and time-consuming methods. The corresponding paragraph has been revised to highlight this aspect.

Comment #7: I don't have access to a RISC-V machine, but I tested out your source code in the gitlab repository on an x86 machine. I think the instructions were easy to follow, but I had some questions:

- ***M_REAL_SIMPLE_F – what optimizations are done here? It's faster than M_REAL_REFERENCE, but slower than M_REAL_SIMPLE.***
- ***What is the difference between GENERIC and SIMPLE? Why is GENERIC slower than simple?***

Answers:

- M_REAL_SIMPLE is the same kernel as M_REAL_REFERENCE, the only difference is that we disabled compiler vectorization in the reference version. M_REAL_SIMPLE_F is the original ELPA code, written in Fortran. In our case, the C compiler seems to be doing a

better job than the Fortran compiler.
- The generic kernel includes several loop unrolls designed to enhance instruction pipelining. This delivers good results when the optimisation levels are reduced and vectorisation is disabled. Our compiler does better optimizations with the simple kernel.

Competing Interests: No competing interests were disclosed.

Reviewer Report 28 August 2024

https://doi.org/10.21956/openreseurope.19799.r42938

© 2024 Wyrzykowski R. This is an open access peer review report distributed under the terms of the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

Roman Wyrzykowski
Czestochowa University of Technology, Częstochowa, Poland

Summary of the article

This paper deals with porting ab initio electronic structure applications to an innovative RISC-V prototype platform. This platform's distinguished feature is a wide Vector Processing Unit (VPU) that can process vectors up to 256 double-precision elements, 32 times larger than current SIMD architectures. In these widely used HPC codes, solving the eigenvalue problem is often the critical computational bottleneck, consuming even up to 90% of the total computational time. Focusing on a mini-application extracted from the ELPA eigensolver library, the de facto standard in the domain, the authors systematically optimize performance targeting RISC-V-based technologies fully developed within the EU.

Details

In the systematic optimization methodology, the authors use three strategies to leverage vector instructions and increase computational efficiency, namely, (i) auto-vectorization, (ii) loop fusion, and (iii) manual vectorization with intrinsic. What is important is that performance is evaluated not only with software emulation but also on an FPGA-based hardware platform, showing outstanding improvements in instruction count and execution cycles.

Among the primary indicators of performance improvements are the following:
1. For the ELPA mini-app executed on the FPGA platform, the auto-vectorization reduces instructions and cycles by a factor of respectively 54.5x and 9.7x, compared to the scalar version.
2. The performance was finally improved in the version with intrinsics, obtaining a speedup of 210x in instructions, and 23.9x in cycles.

The reviewed research demonstrates the massive potential of the co-design approach and RISC-V open architecture for developing future HPC hardware and compilers, which can achieve unprecedented application performance and computational efficiency. This work makes a clear contribution to European strategy in the HPC domain and promotes technologies fully developed within the EU.

Is the work clearly and accurately presented and does it cite the current literature?
Yes

Is the study design appropriate and does the work have academic merit?
Yes

Are sufficient details of methods and analysis provided to allow replication by others?
Yes

If applicable, is the statistical analysis and its interpretation appropriate?
Yes

Are all the source data underlying the results available to ensure full reproducibility?
Yes

Are the conclusions drawn adequately supported by the results?
Yes

Competing Interests: No competing interests were disclosed.

Reviewer Expertise: HPC, computing architectures, parallel algorithns and applications

I confirm that I have read this submission and believe that I have an appropriate level of expertise to confirm that it is of an acceptable scientific standard.

Author Response 23 Oct 2024

julio gutierrez

We are very grateful for your positive feedback on our paper. Your thorough review and encouraging comments are greatly appreciated. We are pleased to know that our work has met your expectations, and we value the time and effort you have invested in reviewing our manuscript.

Competing Interests: No competing interests were disclosed.

Reviewer Report 28 August 2024

https://doi.org/10.21956/openreseurope.19799.r42945

© 2024 Kamaleldin A. This is an open access peer review report distributed under the terms of the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

Ahmed Kamaleldin
Technische Universitat Dresden, Dresden, Germany

This article presents a method to port the ELPA eigensolver to the RSIC-V based vector processor architecture (EPAC-VEC). The architecture is part of the EPI project, it can handle vectors up to 256 double-precision elements.

The execution was performed on a user-space emulator called Vehave for RISC-V V ISA. The article proposal adapts the ELPA mini-app to run on a long vector architecture.

Several compiler autovectorisation techniques are investigated: 1) auto-vectorization, 2) fusing similar loops, and 3) using intrinsics.

The evaluation shows that applying those auto-vectorization techniques can reduce the instruction counts by 13.4x and the total number of clock cycles.

Comments:

1- Adding several applications (as use cases) for evaluation is recommended.

2- Evaluating the energy efficiency and achieved computing performance is recommended.

3- A comparison with other HPC vector processors could be useful for evaluation.

Is the work clearly and accurately presented and does it cite the current literature?
Yes

Is the study design appropriate and does the work have academic merit?
Yes

Are sufficient details of methods and analysis provided to allow replication by others?
Yes

If applicable, is the statistical analysis and its interpretation appropriate?
Yes

Are all the source data underlying the results available to ensure full reproducibility?
Yes

Are the conclusions drawn adequately supported by the results?
Yes

Competing Interests: No competing interests were disclosed.

Reviewer Expertise: System-on-chip, Reconfigurable computing, Multi-core architecture, Heterogeneous compute systems

I confirm that I have read this submission and believe that I have an appropriate level of expertise to confirm that it is of an acceptable scientific standard.

Author Response 23 Oct 2024

julio gutierrez

We thank the referee for his feedback on our paper. Regarding the comments provided, we would like to clarify that:

1. The ELPA library is currently used by several of the most popular electronic structure application packages, some of them cited in the manuscript. The present kernel represents a key section in the diagonalisation of a dense matrix, which is a relevant use case to many of this family of codes. While testing the performance of a full application on the VPU is not possible at the moment, this will definitely be an interesting continuation of this work, which we plan to carry out in the near future.
2. We agree that measuring energy consumption would add insight information to the evaluated technology. However, thorough energy measurements cannot be shared due to NDA constraints enforced by the EPI consortium. We can add that:
    1. The EPAC-VEC test chip is implemented using Global Foundries 22nm transistor technology, which is a fairly outdated chip technology compared with the state-of-the-art chip manufacturing technology. While the test chip development effort still represents a ground-breaking step forward for European technological horizon, the detailed power figures would be at this stage probably not representative.
    2. The EPAC-VEC test chip does not embed a memory controller (it relies on a high-speed SerDes and an external FPGA for memory and I/O operations): this would make the comparison of power drain figures difficult to compare with most state-of-the-art established technologies.
    3. RISC-V is a new, simple and modular ISA used to guarantee a lean implementation (as compared to incremental ISAs such as Intel x86 that need to guarantee support to legacy instruction), theoretically leading to a more energy efficient final solution.
    4. Since the architecture of EPAC-VEC explores a design point that is fairly extreme (using extremely wide vectors), the main goal of our work is to address programmability and performance gain delivered by this architecture.
3. NEC Aurora SX is the only large vector accelerator with an architecture comparable to the one used in EPAC-VEC. While the comparison is definitely possible, the analysis of the results should take into account silicon area considerations. NEC Aurora SX architecture in fact treats the same vector size using 12× more functional units, delivering higher performance at a significant cost in terms of area. Thus, in the interest of space and clarity of the presentation we decided to keep this comparison out of this paper and will consider it for an extension.

Competing Interests: No competing interests were disclosed.

Reviewer Report 21 August 2024

https://doi.org/10.21956/openreseurope.19799.r42941

© 2024 Vella F. This is an open access peer review report distributed under the terms of the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

Flavio Vella
University of Trento, Trento, Italy

The document discusses the integration of **ab initio electronic structure methods** with a RISC-V vector architecture in High-Performance Computing (HPC). Ab initio calculations, particularly in Density Functional Theory (DFT), are computationally intensive, with eigenvalue problems often being the primary bottleneck, consuming up to 90% of total computational time in large systems.

The project **focuses on porting the ELPA eigensolver library to a RISC-V prototype featuring a wide Vector Processing Unit (VPU).** This VPU can handle vectors up to 256 double-precision elements, significantly larger than those managed by conventional SIMD architectures like Intel's AVX-512. The approach involves using a mini-app from the ELPA library to optimize performance on the RISC-V platform.

Details
Three optimization strategies were employed to reduce vector instructions and improve computational efficiency. 1) autovectorisation, 2) loop fusion, and 3)manual vectorisation using intrinsics.

Performance was evaluated using both emulation and FPGA platforms, showing significant reductions in execution cycles and instruction counts.

The work is positioned within the broader context of preparing ab initio methods for exascale computing, where increasing architectural heterogeneity is expected. The co-design approach provides critical feedback for the development of future HPC hardware and compilers, influencing the design of post-exascale systems. This research contributes to the EU's strategic goals in HPC, advancing the readiness of key computational codes for next-generation systems.

Key Performance Indicators

Reduction in Vector Instructions. Initial implementation with autovectorization resulted in 103,784 vector instructions.

1. By suppressing the Q-stripes and leveraging the full vector length of 256 elements, vector instructions were reduced to 17,304, a 6x reduction.

2. Further optimization with loop fusion reduced the vector instructions to 15,405, a 6.7x reduction from the original.
3. Manual vectorization using intrinsics reduced the vector instructions to 9,499, achieving a 13.4x improvement over the initial version.

Performance Improvements. The scalar version required 227,443,896 cycles and 231,667,638 instructions.

1. Autovectorization reduced cycles and instructions by 9.7x and 54.5x, respectively.
2. With fused loops, there was a 9.9x reduction in cycles and a 96.1x reduction in instructions.
3. Using intrinsics, the improvements reached 23.9x in cycles and 210x in instructions compared to the scalar version.

Memory Access Reduction:
The optimization strategies led to significant reductions in memory accesses, with a focus on the costly load (vle) and store (vse) instructions. For example, memory accesses (vle calls) were reduced from 7,040 to 6,064 after loop fusion.

Is the work clearly and accurately presented and does it cite the current literature?
Yes

Is the study design appropriate and does the work have academic merit?
Yes

Are sufficient details of methods and analysis provided to allow replication by others?
Yes

If applicable, is the statistical analysis and its interpretation appropriate?
Yes

Are all the source data underlying the results available to ensure full reproducibility?
Yes

Are the conclusions drawn adequately supported by the results?
Yes

Competing Interests: No competing interests were disclosed.

Reviewer Expertise: High-Performance Computing, GPU Computing, Distributed-memory systems.

I confirm that I have read this submission and believe that I have an appropriate level of expertise to confirm that it is of an acceptable scientific standard.

Author Response 23 Oct 2024

julio gutierrez

We sincerely thank you for your positive feedback on our paper and for taking the time to thoroughly review it.

Competing Interests: No competing interests were disclosed.