RISC-V for Genome Data Analysis: Opportunities and Challenges

Lorién López-Villellas*, Esteve Pineda-Sánchez†, Asaf Badouh†, Santiago Marco-Sola‡†, Pablo Ibáñez*, Jesús Alastruey-Benedé*, Miquel Moretó§†
*Universidad de Zaragoza, Zaragoza, Spain
†Barcelona Supercomputing Center, Barcelona, Spain
‡Universitat Autònoma de Barcelona, Barcelona, Spain
§Universitat Politècnica de Catalunya, Barcelona, Spain

***Abstract*—The RISC-V ISA has gained significant momentum in High-Performance Computing (HPC) research and market due to its open-source nature, fostering collaborative research and innovation. The ever-growing RISC-V-based hardware/software ecosystem has made it an attractive option for HPC application development and production. Within the field of biomedical research, genome data analysis has emerged as a crucial step towards personalized medicine, demanding substantial computational resources and more efficient tools.**

This paper presents a benchmark suite of genome analysis kernels ported to RISC-V and their evaluation on modern RISC-V systems. Our work evaluates the RISC-V toolchain's maturity and the software/hardware ecosystem's readiness for its adoption for genome data analysis. This study aims to provide valuable guidance for researchers and practitioners interested in adopting RISC-V for genome analysis, and provides feedback to the RISC-V community on the challenges that need to be addressed for RISC-V to become an efficient HPC platform.

***Index Terms*—RISC-V, Genome Analysis, Benchmarking, HPC**

I. Introduction

Over the past years, the RISC-V instruction set architecture (ISA) has gained significant momentum and popularity, largely due to its open nature, which allows for free use, modification, and distribution. This characteristic has made it an affordable and accessible alternative to traditional and proprietary ISAs, like Intel's x86 and ARM. RISC-V's modular design allows for easy customization and scalability, making it adaptable to a wide range of applications and computing devices, from low-power devices to large-scale HPC supercomputers. The RISC-V project has garnered support from industry and academia, helping to drive its development forward. Developing RISC-V microprocessors has become a strategic priority for the European Union, as evidenced by many ongoing initiatives and research projects. To ensure competitiveness, it is crucial to consolidate a strong RISC-V community that can foster research and development of RISC-V technology for HPC-demanding applications, including climate modelling [1], molecular simulations [2], machine learning [3], and computational genomics [4].

With the advent of sequencing technologies, genome data analysis tools have gained special relevance in biomedical and healthcare research. Moreover, the wide accessibility to sequencing technologies has enabled the development of personalized medicine [5] for the early detection of diseases, such as hereditary cancers (e.g., Lynch syndrome and breast cancer [6]) and other genetic conditions [7]. As sequencing technologies evolve, the cost of sequencing a human genome drops exponentially, and the current sequencing data production is outpacing Moore's Law. This situation has resulted in an increasing computational bottleneck in the genome analysis pipelines [8]. To face this challenge, an increasing number of algorithms [4] and cutting-edge hardware accelerators [9] have been developed to accelerate the genome analysis pipelines.

This paper describes the porting of the GenArchBench benchmark suite to the RISC-V ISA. GenArchBench is a selection of 13 kernels belonging to widely used genomics tools that cover the most important steps of genome sequencing. We evaluate our port on two RISC-V systems to explore the maturity and suitability of RISC-V core designs for executing genomics applications. Our study aims to help other developers working on RISC-V porting and to guide the designers of RISC-V processors by pointing out the identified shortcomings.

This study makes the following contributions:

- We present a porting of the GenArchBench benchmark suite to the RISC-V architecture, exploiting the RISC-V vector extension when possible.
- We discuss the challenges and limitations encountered during the RISC-V porting, regarding the development toolchain, the availability of libraries and software for RISC-V, and the support level from the ISA, with particular emphasis on the RISC-V vector extension.
- We analyze the performance of GenArchBench on two different RISC-V platforms (Unmatched by SiFive and Allwinner by Alibaba) compared to an Intel Xeon Skylake-based server baseline for reference.

II. Target Hardware Systems

Our study and experimental evaluation involve two RISC-V platforms. Additionally, we include an x86_64 machine for comparison to provide baseline performance for a well-established and widely used architecture in HPC environments. The main characteristics of the systems are shown in Table I and are described throughout this section.

A. HiFive Unmatched

The HiFive **Unmatched** from SiFive is a RISC-V Linux development board released in May 2021. It is powered by the SiFive Freedom U740 (FU740) system-on-chip that operates at 1.2 GHz and features a dual-issue in-order 64-bit execution

Table I
OVERVIEW OF THE MACHINES OF THE EXPERIMENTAL SETUP.

	Unmatched	Allwinner	SKX
CPU	Freedom U740	XuanTie C906	Skylake Platinum 8160
ISA	RV64GC+IMAC	RV64GCV	x86_64
VLEN	—	128 bits	128/256/512 bits
# Cores	4 + 1	1	24
Pipeline	in-order	in-order	out-of-order
Commit	2 instr.	1 instr.	4 μOP
Frequency	1.2 GHz	1 GHz	2.1 GHz
L1I	32 KB	32 KB	32 KB
L1D	32 KB	32 KB	32 KB
L2	—	—	1 MB
LLC	2 MB	—	33 MB
Memory	16 GB DDR4	2 GB DDR3	48 GB DDR4

pipeline. It includes four SiFive U74 compute cores, each with a 32 KB instruction cache and a 32 KB data cache. These cores implement the RV64GC instruction set architecture (G and C extensions). The FU740 also includes one SiFive S71 auxiliary core for real-time applications. This core has a 16 KB instruction cache and an 8 KB data tightly integrated memory (DTIM) and implements the RV64IMAC ISA. The FU740 SoC includes a 2 MB coherent banked L2 cache and 16 GB of DDR4 memory operating at 1866 MT/s.

B. Allwinner D1

The **Allwinner** D1 (D1-H) is a system-on-a-chip (SoC) manufactured by Allwinner and released on April 2021. It is equipped with a single Alibaba T-Head XuanTie C906 core running at 1 GHz. This core implements the RISC-V base 64-bit instruction set architecture (ISA) and the G, C, and V extensions (RV64GCV). Additionally, it implements the RISC-V vector ISA v0.7.1, supporting a vector length of up to 128 bits. The core has an in-order single-issue execution pipeline, a 32 KB L1I, and a 32 KB L1D. The SoC is equipped with 2 GB of DDR3 memory.

C. Intel Xeon Skylake

As part of our experimental setup, we include an x86_64 compute node alongside the RISC-V systems previously described, which serves as a performance reference for production environments. This compute node, which we refer to as **SKX**, is powered by an Intel Xeon Skylake Platinum 8160 processor with 24 cores running at a peak clock speed of 2.1 GHz. Each core has a 32 KB L1I and a 32 KB L1D and is capable of retiring up to 4 micro-operations[1] per cycle. The system also includes a 33 MB last-level cache (LLC). The x86_64 CPU supports SSE, AVX2, and AVX-512 single instruction on multiple data (SIMD) extensions, enabling vector operations on 128, 256, and 512 bits registers, respectively.

III. GENARCHBENCH

The GenArchBench benchmark suite, available at https://github.com/LorienLV/genarchbench, compiles a set of representative kernels belonging to widely-used tools and libraries for genome analysis. It includes ten kernels from the GenomicsBench test suite and three additional kernels: the Bit-Parallel Myers algorithm, the Wavefront Alignment algorithm, and a SIMD-accelerated version of Minimap2's chaining implementation (FAST-CHAIN). Table II shows the name, brief description, use-case, application of origin, and characterization of each kernel from GenArchBench.

The GenArchBench kernels cover a broad range of common steps of genome analysis pipelines. The starting point for many standard genome analysis pipelines is basecalling, i.e., processing the output from sequencing machines. GenArchBench includes the ABEA kernel (from Nanopolish) and the NN-BASE kernel (from Bonito) as representative basecalling kernels. Once the sequenced DNA bases are known, most genome analyses require locating those sequences into a previously known reference genome by performing the read mapping step. Notable applications for read mapping are BWA-MEM2 or Minimap2. GenArchBench includes the BPM, BSW, WFA, CHAIN, FAST-CHAIN, and FMI kernels from this step. For some studies, there is no reference genome to read map against. In those situations, we need to perform a de-novo assembly of the genome before any read mapping can be performed. GenArchBench includes the DBG (Platypus) and KCNT (Flye) kernels, used within standardized and widely used assembly tools. When the sequenced DNA is located in the reference genome, we can detect variations between the sample and the reference genome. This process, known as variant calling, comprises GenArchBench's kernels NN-VARIANT (Clair3) and PILEUP (Medaka).

Additionally, the GenArchBench suite comprises a set of representative input datasets to execute the aforementioned kernels. It is important to note that genome analysis pipelines generally process large datasets, requiring large memory footprints. Alas, the RISC-V machines of our experimental setup are quite modest in memory capacity. Thus, we used GenArchBench's reduced input datasets (small inputs) to enable agile development and reasonable execution tests. Furthermore, we generated tiny inputs for the most memory-intensive kernels (FMI, KCNT, and POA) to enable their porting and execution.

IV. PORTING GENARCHBENCH TO RISC-V

This section presents the outcomes of porting the genomics benchmark suite GenArchBench to RISC-V. We describe the set of kernels successfully ported for RISC-V architectures, the methodology and tools employed, and a primer for accelerating these kernels using RISC-V vector extensions (RVV). Furthermore, we discuss the challenges and limitations encountered during the porting, including the software and toolchain support for RISC-V, the RISC-V ISA suitability for genomics kernels, and other system and hardware caveats.

1) RISC-V GenArchBench Porting: From the initial 13 genomic kernels included in the GenArchBench, we have successfully ported and executed 10 of them. Unfortunately, the lack of RISC-V support from the toolchain, libraries, or hardware has prevented the porting of some kernels. In particular, the lack of RISC-V support from the widely-used Pytorch and Tensorflow libraries prevented the porting of the DL-based kernels (NN-Base and NN-Variant). As will be discussed later, the FAST-CHAIN kernel has been successfully

[1]Instructions of the x86-64 ISA are broken down into simpler operations inside the CPU pipeline.

Table II
KERNELS INCLUDED IN GENARCHBENCH.

Kernel	Description	Use-Case	Genomics Tool	Characterization
ABEA	Adaptive Banded Signal to Event Alignment	Basecalling	Nanopish	Compute-bound
BPM	Bit-Parallel Myers Alignment	Read mapping	GenArchBench	Compute-bound
BSW	Banded Smith-Waterman	Read mapping	BWA-MEM2	Compute-bound
CHAIN	Seed Chaining	Read mapping	Minimap2/GenomicsBench	Compute-bound
FAST-CHAIN	SIMD-enabled Seed Chaining	Read mapping	Minimap2/GenomicsBench	Compute-bound
DBG	De-Bruijn Graph construction	De-novo assembly	Platypus	Compute-bound
FMI	FM-Index	Read mapping	BWA-MEM2	Memory-bound
KCNT	K-mer Counting	De-novo assembly	Flye	Memory-bound
NN-BASE	Neural Network-based Base Calling	Basecalling	Bonito	Compute-bound
NN-VARIANT	Neural Network-based Variant Calling	Variant calling	Clair3	Compute-bound
PILEUP	Pileup Counting	Variant calling	Medaka	Compute-bound
POA	Partial-Order Alignment	De-novo assembly	SPOA	Compute-bound
WFA	Wavefront Alignment Algorithm	Read mapping	AnchorWave	Compute-bound

ported to RISC-V by emulating certain vector instructions. However, we have been unsuccessful in executing the vectorized version of the kernel on Allwinner, the only RISC-V machine with support for the vector extension. FAST-CHAIN is a variant of the CHAIN kernel that has been modified to eliminate all heuristics in order to enable vectorization. Accordingly, assessing its scalar performance is not relevant or informative. Thus, Section V does not include data on FAST-CHAIN.

For the porting, we followed a straightforward development approach based on porting, verifying, and optimizing iteratively. We compiled ground-truth results from the input datasets to ensure the ported kernels behave as expected. Moreover, we used an automatic validation system to test all the ported kernels during development.

As a first step, we focused on the scalar adaptation of the kernels with particular emphasis on validating the results. Then, we proceeded to perform executions on scalar cores and gather profiling information. Afterwards, we worked on extending the porting to vector instructions (RVV) to accelerate the kernels' execution on RVV-enabled processors. Compared to the scalar adaptation, the RVV extensions have little software and hardware support, limiting the development to simulated environments using experimental tools.

2) Software and Libraries Support: Due to its recent introduction, RISC-V-based platforms still lack widespread application and library support. As a result, most applications and libraries require recompilation and careful ad-hoc optimizations to exploit the capabilities of current RISC-V processors. In some cases, these libraries lack specific RISC-V support, tampering the efficient porting of high-performance applications. Notable examples in genome sequence data analysis include libraries for common file-format management (like HDF5 and HTSlib libraries) and Deep Learning (DL) frameworks (like Pytorch and TensorFlow).

Nowadays, Deep Learning (DL) libraries and tools have gained widespread adoption in multiple research areas and industrial applications. In particular, DL-based applications for genome analysis rely on the popular Pytorch and TensorFlow libraries. These DL frameworks depict many libraries' dependencies and lack precompiled versions for RISC-V. Moreover, due to the computationally intensive nature of these libraries, straightforward and non-optimized porting to RISC-V could result in significant execution inefficiencies. For that, many efforts from the HPC community aim to enable high-performance execution of DL-libraries on RISC-V.

3) Development Toolchain Support: To address the challenges of porting and optimising widely-used applications and libraries, it is paramount to have mature and robust toolchain support. We have utilised several development tools for the genomics porting to RISC-V; some are stable and upstream available, others experimental and under active development.

System infrastructure. For the porting development and evaluation, we performed multiple executions on commercial RISC-V scalar cores running standard Linux distributions; i.e., Ubuntu 20.04 (Unmatched), Fedora 33 (Allwinner), SUSE Linux Enterprise Server 12 SP2 (SKX). In addition, we utilised a QEMU virtual machine to emulate a generic 64-bit RISC-V processor for development, profiling, and RVV emulation purposes. Note that the selected QEMU virtual machine does not natively support RVV extension. We utilised the Vehave emulator for vectorial kernels, which enabled us to debug and functionally validate the ported kernels and perform performance profiling and analysis on an emulated environment.

Compilers. We mainly used GCC v10.3.0 and LLVM-based v12.0.0 compilers to compile libraries and tools. Both compilers have shown no difficulties to generate scalar binaries. However, RVV and auto-vectorization support is still under development for mainstream compilers. As a result, compiling vector code has been challenging and error-prone in many situations. For that reason, we resort to BSC-Clang, an extended Clang compiler from the EPI project supporting RVV extensions via auto-vectorization, pragma annotations, and C-Style intrinsics.

Debuggers and profilers. Currently, commonly used tools for profiling and debugging (Valgrind, GNU Debugger (GDB), and Perf) offer limited support for RISC-V. For instance, the RISC-V support disassembling instructions (using RISC-V mnemonics) is a crucial feature when developing applications for RISC-V platforms. Therefore, fully-compliant RISC-V disassembling in GDB debugger is paramount, including instructions on the extended ISA. In that regard, we observe that the llvm-mc tool (Clang toolchain) is more mature, being able to convert opcodes to mnemonics and vice versa. Moreover, we noticed that the widely used Valgrind does not support RISC-V. Due to its importance and usefulness, there is an

ongoing effort to port to RV64GC (https://github.com/petrpavlu/valgrind-riscv64).

Regarding profilers, like perf tools, we found current RISC-V cores lack numerous standard counters mapped on other architectures. In the case of the RISC-V cores studied in this work, we were only able to measure cycles and instructions in Unmatched. The Allwinner does not have counters.

To provide a better understanding of the kernel's performance and guide the development and optimization on RISC-V platforms, profiling utilities like Paraver have been promptly upgraded. Paraver can be used in conjunction with software emulators like Vehave to produce performance traces and analyse them using a versatile GUI. This tool allows analyzing performance metrics (like instructions and register usage), studying the memory access pattern, displaying annotated timelines with events, and more. These tools allowed us to inspect the critical regions and identify potential performance overheads. Moreover, we were able to investigate vectorial kernels without the need for a real vector machine.

4) ISA support: Ratified elemental extensions of the RISC-V architecture, such as the multiplication/division (M), floating-point (F), and double-precision (D) extensions, have been available for more than a decade and are generally adopted by commercial hardware and mainstream compilers. Most kernels ported on this project rely solely on these basic extensions, allowing its straightforward compilation and execution on any compliant RISC-V core. However, some RISC-V cores lack support for specific instructions from the specification. For example, the Allwinner machine does not support the optional `fence.tso` instruction used by the POA kernel.

Porting the genomic kernels to the RISC-V vector (RVV) extension has proven challenging in many cases. We found that only one kernel could be partially vectorized automatically by the compiler (i.e., WFA). For the kernels BSW and CHAIN, we develop a vectorial implementation based on RVV-intrinsics. For the remaining kernels, we found that the current RVV extension lacks essential instructions used by genomic kernels.

In particular, we found that, unlike AVX-512 (x86_64) or SVE (Arm) ISAs, the well-known count leading zeros (`clz`) and count trailing zeros (`ctz`) instructions lack a vector counterpart in the RVV specification, useful for the CHAIN and WFA kernels. The scalar versions of `clz` and `ctz` are part of the ratified bit manipulation extension (B), and their vector versions are currently part of the cryptography extension proposal. Beyond genomic kernels, `clz` and `ctz` are useful for multiple applications, including square root computation, Huffman decoding, binary logarithm computation, and hash tables indexing. Although their functionality can be emulated by executing other arithmetic instructions, the overall instruction overhead has a non-negligible toll on performance.

Similarly, the RVV specification lacks specific support to reverse the order of the elements within a vector register, a helpful operation for some genomic kernels. Vector reverse can be conveniently implemented using the `vid` instruction to compute an index vector and the `vrgather` instruction to generate the final result. However, some additional instructions are needed to rearrange the indices produced by `vid`. We claim that a more powerful version of the `vid` instruction is possible. If RVV supported an instruction similar to `vid`, allowing to specify the starting, ending, and increment values (akin to the functionality of Linux's `seq` command), we could reverse a vector using fewer instructions. Additionally, this new instruction would be useful in a wide range of scenarios, since `vid` usually requires to be paired with additional instructions to compute the desired vector of indices.

In practice, we have encountered issues executing vector kernels on Allwinner, the only RVV-enabled core. Although this processor implements the RVV extension v0.7, it may not execute correctly in certain scenarios. For example, it throws an illegal instruction error when executing the `vmv` instruction, which is used on the kernels BSW, CHAIN, and WFA. Upon further investigation, we found that these kernels used a 64-bit element width (SEW=64), and Allwinner does not support RVV instructions for 64-bit element widths, resulting in an illegal instruction exception. Although the kernels could be adapted to use a smaller element width, the compiler emits 64-bit element width by default. Another example is the execution of RVV gather instructions, required by some kernels like WFA, which also result in a run-time error when executed on Allwinner.

It is important to acknowledge that the RVV specification and cores implementing it are still in the process of being refined and made more robust with future developments. A clear example of this is the specification of the masked instructions, which require using register v0 for the mask operand, although a mask can be stored in any register. Using RVV v0.7, moving a mask register to v0 required several instructions. However, with the introduction of RVV 1.0, this can be achieved with a single instruction. Although the scalar specification and cores prove to be more mature and stable, the RVV specification and hardware are rapidly evolving to be competitive with well-establish vector extensions in the industry like AVX and SVE.

5) Platform Limitations: In the exploration of the maturity and limitations of RISC-V processors to execute genomic applications, we have encountered other technical challenges. Most notably, genomic kernels usually process big amounts of data, requiring large memory footprints. Considering the memory available on current RISC-V platforms, we found that real executions at genome-scale are not feasible as of today. Specifically, Allwinner is with only 2 GB of memory. Considering that representative genomics datasets and databases are usually of the order of tens of gigabytes, we were forced to use reduced inputs for testing the execution of the ported kernels. In some cases, we even had to produce tiny datasets to benchmark the most memory-intensive kernels as described in Section III.

V. Experimental Evaluation

In this section, we evaluate the performance of the two RISC-V machines presented in Section II when executing GenArchBench. First, we present the execution time results for scalar and single-thread executions performed in the RISC-V and SKX machines. Next, we present a scalability analysis for scalar multi-threaded executions on the SiFive Unmatched machine. Due to the challenges and limitations discussed in Section IV, we have not included results for the RVV versions of the kernels.

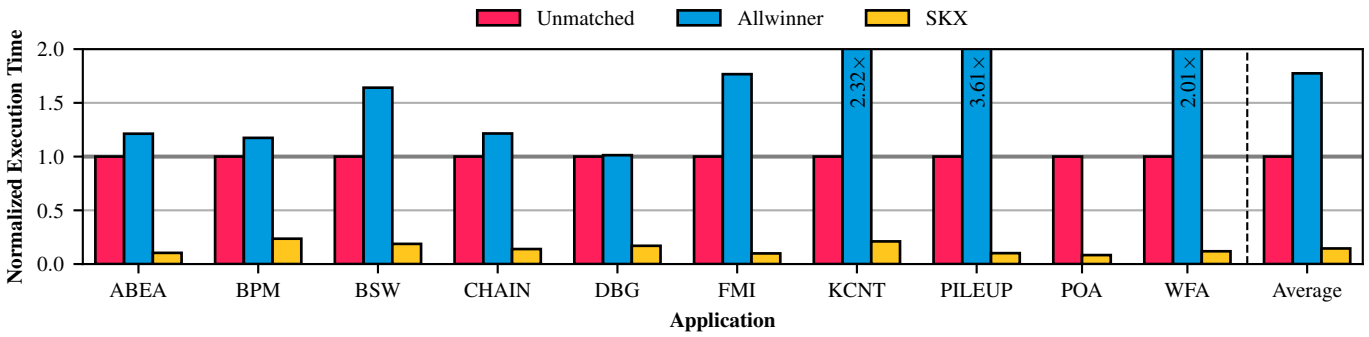

Figure 1. Single-thread and scalar execution time of GenArchBench's kernels. The results are normalized to the performance on the Unmatched machine.

A. Single-thread scalar execution

Figure 1 shows the execution time of GenArchBench's kernels when performing scalar and single-thread executions on the two RISC-V machines, Unmatched and Allwinner, and the reference x86_64 machine, SKX (see Table I). The results have been normalized to the performance on Unmatched. We selected the Unmatched for normalization reference as it is the only RISC-V machine that can successfully execute the scalar version of all the kernels.

We observe that, compared to the SKX, Unmatched shows slowdowns between 3.8× and 12×, for KCNT and POA, respectively. The average slowdown of Unmatched compared to SKX is 6.6×. This is an expected result considering the characteristics of both processors. In particular, SKX works at a higher frequency (1.75×), has a higher superscalar degree (2×), and implements an out-of-order pipeline while Unmatched implements an in-order core. In addition, SKX has significantly larger L2 and L3 caches (16×) and includes hardware prefetchers in L1 and L2 while Unmatched does not implement any prefetcher. These features of their memory hierarchies can make a big difference in some applications.

Moreover, we noticed that Allwinner is consistently slower than Unmatched for all kernels, showing slowdowns ranging between 1.01× and 3.6×. These results are consistent with the characteristics of the machines, as Unmatched operates at a higher frequency (×1.2) and implements a superscalar core (×2). Furthermore, the memory hierarchy may significantly affect the performance of some kernels, considering that Allwinner has only one cache level, divided into L1I and LID (32 KB each).

It would be noteworthy to perform a more detailed analysis to understand the influence of the memory hierarchy on the results. However, this analysis is not possible due to the lack of the necessary hardware counters in the RISC-V systems.

B. Multi-thread scalar execution

The Unmatched processor includes 4 cores, allowing multi-threaded executions using up to 4 threads. Figure 2 shows the speedup of each kernel executing with 2 and 4 threads compared to single-thread execution. Most kernels exhibit near-perfect scalability as the number of threads increases. In particular, 9 of the 10 kernels achieve speedups above 1.85×, using two threads, and above 3.5×, using four threads. The WFA kernel presents the worst scaling behaviour, showing speedups of 1.7× and 2.8× for 2 and 4 threads, respectively.

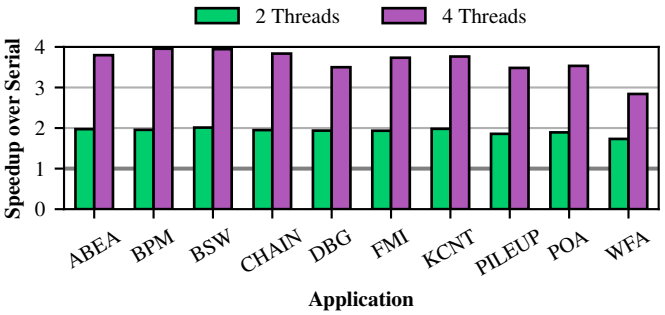

Figure 2. Speedup over serial execution of GenArchBench's kernels on the Unmatched machine. We show the achieved speedup using 2 and 4 threads.

VI. Related Work

The RISC-V architecture has gained significant traction in recent years. As a result, many efforts have been to extend its toolchain and improve RISC-V core design. Moreover, there are several ongoing European projects targeting the design of RISC-V processors. The European Processor Initiative (EPI) aims to design RISC-V processors for HPC. EPAC is the first chip designed as part of the EPI project, combining several domain-specific accelerators with a dual-issue core and a vector processing unit. The eProcessor project is aimed to deliver the first fully open-source European full-stack ecosystem based on a RISC-V CPU and multiple diverse accelerators. The DRAC project targets to design, verify, implement, and manufacture a RISC-V high-performance processor, incorporating different accelerators with applications to security, autonomous navigation, and genomics [10].

Regarding the RISC-V toolchain, we can find studies comparing different RISC-V compilers, such as the work by Poorhosseini et al. [11], which examines the performance and size of binaries generated by GCC and LLVM, and compares the performance of the compilers themselves. Other studies, such as the one by Adit and Sampson [12], have focused on compiler auto-vectorization. We observe there is an increasing interest in extending the RISC-V support to compilers, such as the JIT compiler [13].

Moreover, we find surveys on the RISC-V specifications [14] and many proposals to speed up critical kernels in the genomics field. For instance, Sargantana [15] is a RISC-V processor with support for a subset of the instructions part of the RISC-V vector extension. Additionally, it includes custom vector instructions to speed up the WFA kernel studied in this project. Furthermore, we find other accelerator proposals for genomics based on the RISC-V ISA [16], [17]. Similarly, we can find works leveraging the RVV extensions to speedup critical kernels in different domains, such as the post-quantum cryptography [18] or deep-learning [19]. Ramírez et al. [20] present a vector benchmark suite for RISC-V. Further, we find several studies benchmarking the performance of HPC applications across different RISC-V cores [21], [22] and different architectures [23].

VII. Conclusions

RISC-V's open nature and unique characteristics make it a highly promising ISA for future computing systems. Despite recent advancements, current RISC-V development lacks critical support from tools, applications, libraries, and hardware. In particular, compilers do not offer full support for some of the latest ratified extensions (including RVV), and development tools such as debuggers and profilers offer limited functionality on RISC-V platforms. Moreover, critical libraries (PyTorch and TensorFlow) lack an efficient port to RISC-V. Additionally, RISC-V machines do not provide reliable hardware support for some ISA extensions and offer limited support for hardware counters. Currently, RISC-V core designs are constrained in resources and not mature enough to compete with established architectures such as x86_64 or Arm.

The extensibility and openness of RISC-V present vast opportunities for developing new domain-specific extensions and accelerators. In this regard, leveraging vector instructions represents a significant leap towards achieving better performance and energy efficiency. Thus, future HPC RISC-V designs require support for the recently ratified vector extension. Notwithstanding the several challenges that need to be addressed, RISC-V has the potential to foster a powerful ecosystem for HPC computing, in general, and genome data analysis, in particular.

Acknowledgment

This work has been partially supported by the Spanish Ministry of Science and Innovation MCIN/AEI/10.13039/501100011033 (contracts PID2019-107255GB-C21 and PID2019-105660RB-C21), by the Generalitat de Catalunya (contract 2017-SGR-1328), by the Gobierno de Aragón (T58_23R research group), and by Lenovo-BSC Contract-Framework Contract (2020). Santiago Marco was supported by the Agencia Estatal de Investigación (Spain) under Juan de la Cierva fellowship grant IJC2020-045916-I. Miquel Moretó was partially supported by the Agencia Estatal de Investigación (Spain) under Ramón y Cajal fellowship number RYC-2016-21104.

We want to thank Roger Ferrer, Kilian Peiro, and Pablo Vizcaino for all of their useful help.

References

[1] W. M. Washington *et al.*, "The computational future for climate and earth system models: on the path to petaflop and beyond," *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, vol. 367, no. 1890, pp. 833–846, 2009.

[2] G. Martinez-Rosell *et al.*, "Drug discovery and molecular dynamics: methods, applications and perspective beyond the second timescale," *Current topics in medicinal chemistry*, vol. 17, no. 23, pp. 2617–2625, 2017.

[3] M. S. Louis *et al.*, "Towards deep learning using tensorflow lite on risc-v," in *Third Workshop on Computer Architecture Research with RISC-V (CARRV)*, vol. 1, 2019, p. 6.

[4] M. Alser *et al.*, "From molecules to genomic variations: Accelerating genome analysis via intelligent algorithms and architectures," *Computational and Structural Biotechnology Journal*, vol. 20, pp. 4579–4599, 2022.

[5] H. K. Brittain *et al.*, "The rise of the genome and personalised medicine," *Clinical Medicine*, vol. 17, no. 6, p. 545, 2017.

[6] R. F. Green *et al.*, "Evaluating the role of public health in implementation of genomics-related recommendations: a case study of hereditary cancers using the cdc science impact framework," *Genetics in Medicine*, vol. 21, no. 1, pp. 28–37, 2019.

[7] D. P. Germain *et al.*, "The benefits and challenges of family genetic testing in rare genetic diseases—lessons from fabry disease," *Molecular Genetics & Genomic Medicine*, vol. 9, no. 5, p. e1666, 2021.

[8] G. Lightbody *et al.*, "Review of applications of high-throughput sequencing in personalized medicine: barriers and facilitators of future progress in research and clinical application," *Briefings in Bioinformatics*, vol. 20, no. 5, pp. 1795–1811, Jun. 2019.

[9] T. Robinson *et al.*, "Hardware acceleration of genomics data analysis: challenges and opportunities," *Bioinformatics*, vol. 37, no. 13, pp. 1785–1795, 05 2021.

[10] J. Abella *et al.*, "An academic risc-v silicon implementation based on open-source components," in *2020 XXXV Conference on Design of Circuits and Integrated Systems (DCIS)*. IEEE, 2020, pp. 1–6.

[11] M. Poorhosseini *et al.*, "A compiler comparison in the RISC-v ecosystem," in *2020 International Conference on Omni-layer Intelligent Systems (COINS)*. IEEE, aug 2020.

[12] N. Adit *et al.*, "Performance left on the table: An evaluation of compiler autovectorization for RISC-v," *IEEE Micro*, vol. 42, no. 5, pp. 41–48, sep 2022.

[13] Q. Ducasse *et al.*, "Porting a JIT compiler to RISC-v: Challenges and opportunities," in *Proceedings of the 19th International Conference on Managed Programming Languages and Runtimes*. ACM, Sep. 2022.

[14] E. Cui *et al.*, "Risc-v instruction set architecture extensions: A survey," *IEEE Access*, vol. 11, pp. 24 696–24 711, 2023.

[15] V. Soria-Pardos *et al.*, "Sargantana: A 1 ghz+ in-order risc-v processor with simd vector extensions in 22nm fd-soi," in *2022 25th Euromicro Conference on Digital System Design (DSD)*, 2022, pp. 254–261.

[16] Z. Wu *et al.*, "An fpga implementation of a portable dna sequencing device based on risc-v," in *2022 20th IEEE Interregional NEWCAS Conference (NEWCAS)*, 2022, pp. 417–420.

[17] L. D. Tucci *et al.*, "Salsa: A domain specific architecture for sequence alignment," in *2020 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, 2020, pp. 147–150.

[18] S. Pircher *et al.*, "Exploring the risc-v vector extension for the classic mceliece post-quantum cryptosystem," in *2021 22nd International Symposium on Quality Electronic Design (ISQED)*, 2021, pp. 401–407.

[19] M. Cococcioni *et al.*, "Vectorizing posit operations on RISC-v for faster deep neural networks: experiments and comparison with ARM SVE," *Neural Computing and Applications*, vol. 33, no. 16, pp. 10 575–10 585, Feb. 2021.

[20] C. Ramírez *et al.*, "A risc-v simulator and benchmark suite for designing and evaluating vector architectures," *ACM Trans. Archit. Code Optim.*, vol. 17, no. 4, nov 2020.

[21] A. Dörflinger *et al.*, "A comparative survey of open-source application-class risc-v processor implementations," in *Proceedings of the 18th ACM International Conference on Computing Frontiers*, ser. CF '21. New York, NY, USA: Association for Computing Machinery, 2021, p. 12–20.

[22] A. Sarihi *et al.*, "Performance evaluation of an out-of-order risc-v cpu: A spec int 2017 study," in *2022 IEEE International Performance, Computing, and Communications Conference (IPCCC)*, 2022, pp. 418–419.

[23] Y. Liu *et al.*, "Performance evaluation of various risc processor systems: A case study on arm, mips and risc-v," in *Cloud Computing – CLOUD 2021*, K. Ye *et al.*, Eds. Cham: Springer International Publishing, 2022, pp. 61–74.