Functional Validation of the RISC-V Unlimited Vector Extension

Ana Fernandes, Luís Crespo, *Student Member, IEEE,* Nuno Neves, *Member, IEEE,* Pedro Tomás, *Senior Member, IEEE,* Nuno Roma, *Senior Member, IEEE,* and Gabriel Falcao, *Senior Member, IEEE*

***Abstract*—Data streaming and data-flow computing paradigms have been on the rise, aiming to improve the performance of general-purpose processors. However, providing support for data streaming typically requires the definition of new Instruction Set Architecture (ISA) extensions, which must be thoroughly validated before being implemented in hardware. This step is usually carried out using Instruction Set Simulators (ISSs), to which the necessary streaming support must be added. Accordingly, this work proposes a new validation simulator for the recently presented stream-based RISC-V ISA Unlimited Vector Extension (UVE). The proposed tool is based on *Spike*, the golden reference ISS for RISC-V extensions. It is capable of processing a wide range of memory access patterns and provides the necessary mechanisms to validate the target extension, as well as to evaluate the resulting instruction reduction gains.**

***Index Terms*—Data Streaming, RISC-V, Instruction Set Simulator, ISA SIMD Extensions, Unlimited Vector Extension.**

I. Introduction

With the end of Dennard Scaling and the presumed slowdown of Moore's Law, traditional methods to improve the processor's performance, such as increasing the clock frequency, have gradually been revealed to no longer be sufficient. Data streaming is a promising paradigm, offering decoupled memory accesses, indexing-free loops, simplified vectorisation, and implicit load and store operations [1]. It works by allowing for the configuration of memory access patterns at the loop preamble, hence allowing data fetching to operate in the background, decreasing the average memory access latency and increasing throughput. Because memory access patterns in most memory-bound applications are mathematically deterministic and thus appropriate for data streaming [2], data transfers can be offloaded to specialised modules, improving throughput and energy efficiency [3], [4].

Several existing data streaming solutions rely on dedicated instructions, as well as on specific streaming modules, leading to the definition of new Instruction Set Architecture (ISA) extensions [3], [5], [6], [2], [7]. Nevertheless, the development of new ISA extensions is not trivial and requires thorough testing and validation before any real hardware implementation attempts are made. For this purpose, Instruction Set Simulators (ISSs) are commonly used [8].

A. Fernandes and G. Falcao are with Instituto de Telecomunicações, University of Coimbra, 3030-290 Coimbra, Portugal (e-mail: {ana.fernandes, gff}@co.it.pt).

L. Crespo, N. Neves, P. Tomás and N. Roma are with INESC-ID, Instituto Superior Técnico, University of Lisbon, 1000-029 Lisboa, Portugal (e-mail: luis.miguel.crespo@tecnico.ulisboa.pt; {nuno.neves, pedro.tomas, nuno.roma}@inesc-id.pt).

Because most recent data streaming extensions, including the Unlimited Vector Extension (UVE), adopt RISC-V as their base ISA, this is the target architecture for the herein proposed simulation tool. However, there is a tough compromise between simulation accuracy and speed when choosing between the available RISC-V simulators [9]. In particular, while QEMU is a generic tool that allows the modelling of different architectures, its complexity hinders the implementation of new extensions. On the other hand, *Spike*[1] is the golden reference for validating RISC-V extensions, and is widely used as the proof-of-concept target [10]. As such, it was chosen as the most appropriate simulator for the development of the aimed tool.

In accordance, this work presents a new simulation and validation tool to support the development of RISC-V data streaming Unlimited Vector Extension (UVE). For this purpose, data streaming mechanisms were introduced on *Spike*, on which new stream support instructions were added and tested. By creating such a functional validation tool, it is possible to focus solely on the instructions' behaviour, detaching the ISA development from implementation details, which can be prone to specification errors. Accordingly, the simulator was modified and expanded to include a new Streaming Unit (SU), which receives the statically generated stream descriptors (e.g., by the compiler), with the required memory access patterns, and acts as an address generator. The developed simulator was then used to validate and evaluate the performance of the target streaming extension, by comparing the number of executed instructions of UVE code against scalar code without data streaming. In the sequence of the development of this tool, this extension was meanwhile modified and improved. The developed tool and supporting documentation are publicly available online[2].

II. The Unlimited Vector Extension

This work presents a new simulation environment for the RISC-V Unlimited Vector Extension (UVE) [1], which is here summarised.

A. Overview

UVE is a RISC-V extension born from the combination of Single Instruction, Multiple Data (SIMD) computing and data streaming paradigms [1]. It features a scalable ISA extension,

[1]https://github.com/riscv-software-src/riscv-isa-sim

[2]https://github.com/hpc-ulisboa/UVE2

which includes stream configuration instructions that allow its streaming specification to be resolved at runtime, as well as the pre-loading of input data.

With a total of 474 instructions (including variants), UVE introduces 32 vector registers to the base ISA (named from *"u0"* to *"u31"*). The length of each vector is implementation-dependent, with a minimum value equal to the maximum width (64 bits) of the supported data types (*byte*, *half-word*, *word*, and *double-word*). The maximum value is unlimited at ISA level. Each vector register can be associated with a data stream. In addition, sixteen predicate registers are present, named *"p0"* to *"p15"*, although only eight can be used in arithmetic and regular memory instructions (*p0-p7*). A predicate register corresponds to a vector with a fixed size of 64 *bytes*, and a predicate is thus evaluated according to the data type of the instruction source operands. As a result, in each predicated instruction the predicate register is read for each active lane, and the operation is only performed if it evaluates to 1, as stated by the ISA specification [1]. Register *p0* is hardwired to 1 (predicate true), which means it can be used in operations where predication is not necessary (i.e. non-conditional loops), as all valid lanes of the operating streams execute. The remaining predicate registers (*p8-p15*) are used in the configuration of the other eight.

B. Pattern Representation

In the UVE extension, a *stream* is defined as a predictable vector of data elements processed sequentially. Each stream data element is subject to the same set of operations and is discarded after the computation is complete. This scheme follows the premise that data accesses are often deterministic and, as such, the order in which the data is consumed can be specified beforehand (e.g., by the compiler) [11], [1], [2], [3]. In particular, UVE considers that any regular *n*-dimensional memory access sequence can be represented by the following affine function:

$$y(X) = y_{base} + \sum_{k=1}^{dim_y} x_k \times S_k, \tag{1}$$

with $X = x_1, ..., x_{dim_y}$ and $x_k \in [O_k, E_k + O_k]$, i.e., a stream access $y(X)$ is described as the sum of the base address of an *n*-dimensional variable (y_{base}) with dim_y indexing variables (x_k) multiplied by their respective strides (S_k), each k corresponding to a dimension of the pattern. E_k represents the number of elements in each k dimension and O_k the indexing offset. Naturally, the composition of several functions allows the representation of highly complex patterns. As an example, memory access indirection can be represented by assigning data obtained from the addresses generated by one function to the input variables of another function.

UVE encodes the function parameters with a header (specifying base address, data type, and vectorisation), followed by a list of one-dimensional pattern descriptors ($\mathtt{d}_1$,$\mathtt{d}_2$,$\cdots$) and modifiers ($\mathtt{m}_1$,$\mathtt{m}_2$,$\cdots$). The combination of multiple one-dimensional descriptors allows the description of multi-dimensional patterns, thus resolving Equation (1). Modifiers are used to change the parameters of a dimension descriptor at runtime, allowing the description of more complex memory access patterns. In particular, modifiers change a given target parameter by a fixed value (*static modifiers* - see Figure 1(a)) or by a value obtained from another stream (*dynamic modifiers*). In the last case, this results in native indirect memory access support, which is further enhanced with dedicated scatter-gather accesses (*scatter-gather modifiers* - e.g., $B[A[i]]$, as illustrated in Figure 1(b)).

Listing 1 shows how this representation model is translated into UVE stream configuration instructions, for the particular example of a lower triangular matrix.

III. Proposed Modifications of the Spike Simulator

The base Instruction Set Simulator (ISS), *Spike*, is currently at Version 1.1.0 and supports many RISC-V ISA features. However, data streaming support had yet to be developed for this simulator. For this purpose, several structures were added to the *Spike* simulator. The modified simulator is hereby described in detail and its structure is represented in Figure 2.

The focal component of the proposed simulator is the ***Streaming Unit (SU)***, a new class that has access to the new stream registers. Each register may or may not be associated with a stream, and this module is responsible for the implicit loading and storing of data, as well as the iteration

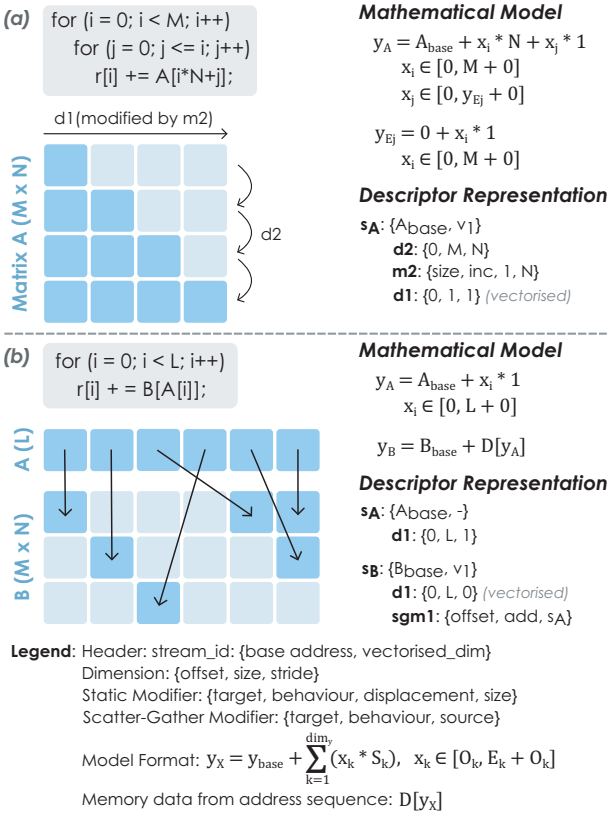

Fig. 1: UVE data stream model and description examples of (a) a triangular access pattern description, where a static modifier is applied to increment the size of the first dimension, and (b) an indirect access pattern description, where a scatter-gather modifier is applied to add the correct matrix index from A to the offset of the dimension of matrix B.

```
1  ; Stream configuration
2  ss.sta.ld.w.v.1       u1, &A ; matrix A stream
3  ss.app                u1, 0, M, N ; d2
4  ss.app.mod.siz.inc.1  u1, 1       ; m2
5  ss.end                u1, 0, 1, 1 ; d1
6
7  ss.sta.st.w  u2, &r      ; vector r stream
8  ss.end       u2, 0, N, 1 ; d1
9
10 ; Computation
11 .iloop:
12     li a1, 0
13     .jloop:
14         so.a.adds.acc.sg a1, u1, p0
15         so.b.ndc.1       u1, .jloop
16     so.v.mvsv.w u2, a1
17     so.b.nc     u1, .iloop
```

Listing 1: Lower triangular matrix row accumulation UVE pseudo-assembly code, as described in Figure 1(a).

of the streams. The iteration and address generation emulate a *streaming engine*, and are implemented in the *descriptor* classes, ***Dimension*** and ***Modifier***, which contain the UVE pattern descriptors. Each stream register, when associated with a stream, is therefore also associated with n dimensions. The SU uses these descriptors to perform the computations described by Equation (1). For the desired functional evaluation, scheduling modules and FIFOs were not required, as streams are iterated while they are being consumed, with each instruction triggering the iteration of the source streams (implicit loading) and the destination streams (implicit storing). Hence, the resulting elements are immediately placed in the associated registers. Additionally, the SU is responsible for keeping track of the stream state, such as the current iteration, and associated flags, as well as the configuration of their memory access patterns. This information is kept in a ***Stream Table***, which can be accessed by some instructions. One key feature of UVE is the flag-dependent loop control. During the iteration of a stream, *End Of Dimension* flags are updated and saved to the *Stream Table*. These are used by branch instructions, with no need for indexing variable control. Instead, loops are controlled by the configured pattern dimensions and their status, as depicted in Listing 1, where loop iteration is determined by the end of the first dimension (line 15) and stream completion (line 17). For simplicity, predication support was developed at the instruction level, meaning that the predicate values are not handled by the SU.

With this streaming infrastructure in place, specialised instructions were added to create, configure and manipulate the

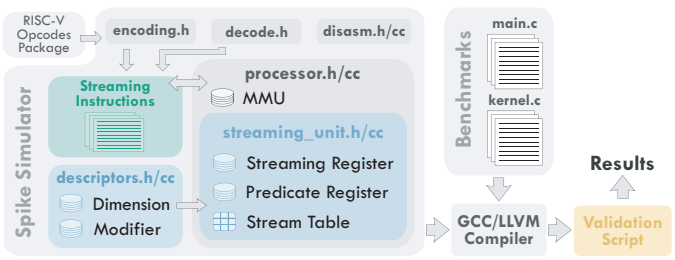

Fig. 2: Simulation environment structure and file organisation.

TABLE I: Benchmarks used for testing and respective characteristics.

	Benchmark	Num. of Streams	Num. of Kernels*	Max. Loop Nesting	Memory Access Pattern
Memory	**Memcpy**	1	1	1	1D
	Stream	10	4	2	2D
BLAS	**SAXPY**	3	1	1	1D
	GEMM	6	2	3	3D
Algebra	**3MM**	3	3	3	3D
	MVT	8	2	2	2D
	GEMVER	17	4	2	2D
	Trisolv	5	1	2	2D + SM
Stencil	**Jacobi-1D**	8	2	1	1D
	Jacobi-2D	12	2	2	2D
ML/AI	**Convolution**	10	1	1	2D
	SGD	9	3	3	3D
Data Mining	**Covariance**	9	3	3	4D + 2 SM
	SpMV-1	4	1	2	3D + DM
	SpMV-2	6	1	2	2D + SG

* The number of kernels corresponds to the number of disjunct loop statements (i.e., excluding nested loops).

streams. For the simulator to recognise these new instructions, the encoding file had to be updated. To obtain the necessary code, the official RISC-V Opcodes project[3] was used, where the encoding of each instruction was added to the standard ISA. For this purpose, the necessary streaming registers and operand encoding were also added to this tool. Moreover, the simulator's *disassembler* was extended to fully support the added instructions, allowing the correct trace generation and debugging of the streaming ISA.

IV. Experimental Validation

Data streaming support was successfully added to *Spike*, as well as most instructions from the UVE data streaming extension. The conceived tool currently supports multi-dimensional pattern descriptors, as well as static, dynamic, and scatter-gather modifiers. Stream-based branching and predication are also supported, as well as multiple arithmetic and vector operations on the streaming registers. In total, 158 instructions have been implemented and validated on *Spike*.

To validate the UVE ISA functional simulation, a comprehensive set of benchmarks from a wide range of application domains was chosen, as depicted in Table I. These were either hand-coded to obtain its corresponding UVE implementation or generated by an adapted and preliminary version of the LLVM compiler with stream-based autovectorisation support (that is also under development [4]). To obtain a meaningful evaluation of the UVE ISA, each benchmark was compared to its scalar version.

Figure 3 depicts the improvement of the number of executed instructions in double-precision floating-point benchmarks. It is possible to observe that UVE achieves a reduction of the number of executed instructions in every benchmark by over 90% in most cases.

By using the newly developed simulation framework, the UVE extension also went through a rigorous validation process

[3]https://github.com/riscv/riscv-opcodes

and was improved in many aspects, therefore validating the usefulness of the proposed tool. In short, UVE received several updates and new features:

- *Added support for scalar streams*: As not all code can be vectorised, the revised version of UVE introduces a flag (in the header instruction) to distinguish between scalar and vector data streams;
- *Revised predication mechanisms*: Instruction predication policies are now directly configured in the predication registers. Additionally, the destination vector register elements corresponding to inactive lanes are now handled with either a *zeroing* (set to zero) or *merging* (retain original value) policy. Stream predication policies were also extended by allowing configuration through the stream description (header).
- *Formalised support for scatter-gather memory accesses*: Conflicting behaviour in using dynamic modifiers required the redefinition of this modifier class. In particular, specific scatter-gather modifiers were added to differentiate between applying an indirect offset to each address generated by a descriptor (i.e., scatter-gather) and modifying a descriptor field with an indirect value (equivalent behaviour to a static modifier);
- *Improved instruction set encoding*: Following the removal of some instructions (in the meanwhile shown to be unnecessary) and simplification of others, the general encoding of instructions was revised.
- *Multiple modifiers per dimension*: The limit of one modifier per dimension was extended to three. This change provides support for more complex kernels, such as *covariance*, which requires modifiers for both the *offset* and *size*, or triangular patterns with indirection, which require both static and dynamic modifiers.

These changes are now reflected in the newly devised UVE specification[4], and are supported by the proposed simulator.

V. Conclusion

In this paper, a new validation and simulation tool for the stream-specialised UVE ISA based on the *Spike* RISC-V simulator is proposed. This new tool provides efficient development and functional evaluation means, not only of the newly introduced instructions but also of their supporting microarchitecture. The new UVE instructions that were added support data streaming with implicit loads/stores, predication, and n-dimensional pattern description with static and dynamic modifiers. Additionally, the simulator's debugging and trace generation tool was extended to fully support the extension. A representative set of benchmarks was tested and verified. This tool can also be used to assess the number of executed instructions, which revealed an average reduction of 92.95% of retired instructions, showing the potential of this data streaming ISA extension.

The modified simulator can easily be adapted to support other data streaming extensions, as their basic functioning is similar to UVE.

[4]https://github.com/hpc-ulisboa/UVE2/blob/main/documentation.pdf

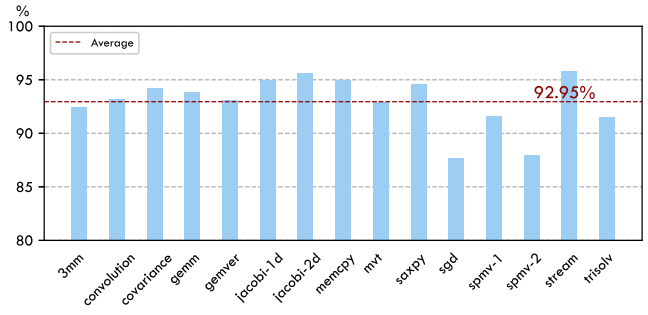

Fig. 3: UVE reduction of retired instructions, relative to scalar code $\left(1 - \frac{Inst_{UVE}}{Inst_{scalar}}\right)$.

Acknowledgment

This work was funded by Fundação para a Ciência e a Tecnologia and Instituto de Telecomunicações under project UIDB/50008/2020 (DOI: 10.54499/UIDB/50008/2020), and also supported by projects UIDB/50021/2020 (DOI: 10.54499/UIDB/50021/2020), 2022.06780.PTDC (DOI: 10.54499/2022.06780.PTDC), and grant 2022.11626.BD.

References

[1] J. M. Domingos, N. Neves, N. Roma, and P. Tomás, "Unlimited Vector Extension with Data Streaming Support," in *2021 ACM/IEEE 48th Annual International Symposium on Computer Architecture (ISCA)*. Valencia, Spain: IEEE, 6 2021, pp. 209–222. [Online]. Available: https://ieeexplore.ieee.org/document/9499750/

[2] Z. Wang and T. Nowatzki, "Stream-Based Memory Access Specialization for General Purpose Processors," in *Proceedings of the 46th International Symposium on Computer Architecture*, ser. ISCA '19. New York, NY, USA: Association for Computing Machinery, 2019, pp. 736–749. [Online]. Available: https://doi.org/10.1145/3307650.3322229

[3] F. Schuiki, F. Zaruba, T. Hoefler, and L. Benini, "Stream Semantic Registers: A Lightweight RISC-V ISA Extension Achieving Full Compute Utilization in Single-Issue Cores," *IEEE Transactions on Computers*, vol. 70, no. 2, pp. 212–227, 2021.

[4] N. Neves, J. M. Domingos, N. Roma, P. Tomás, and G. Falcao, "Compiling for Vector Extensions With Stream-Based Specialization," *IEEE Micro*, vol. 42, no. 5, pp. 49–58, 9 2022.

[5] P. Scheffler, F. Zaruba, F. Schuiki, T. Hoefler, and L. Benini, "Indirection Stream Semantic Register Architecture for Efficient Sparse-Dense Linear Algebra," in *2021 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2021, pp. 1787–1792.

[6] T. Nowatzki, V. Gangadhar, N. Ardalani, and K. Sankaralingam, "Stream-Dataflow Acceleration," in *Proceedings of the 44th Annual International Symposium on Computer Architecture*, ser. ISCA '17. New York, NY, USA: Association for Computing Machinery, 2017, p. 416–429. [Online]. Available: https://doi.org/10.1145/3079856.3080255

[7] Z. Wang, J. Weng, S. Liu, and T. Nowatzki, "Near-Stream Computing: General and Transparent Near-Cache Acceleration," in *2022 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, 2022, pp. 331–345.

[8] M. Garcia, E. Francesquini, R. Azevedo, and S. Rigo, "Hybridverifier: A cross-platform verification framework for instruction set simulators," *IEEE Embedded Systems Letters*, vol. 9, no. 2, pp. 25–28, 2017.

[9] A. Roelke and M. R. Stan, "RISC5: Implementing the RISC-V ISA in gem5," *First Workshop on Computer Architecture Research with RISC-V (CARRV)*, 2017.

[10] H. Genc, S. Kim, A. Amid, A. Haj-Ali, V. Iyer, P. Prakash, J. Zhao, D. Grubb, H. Liew, H. Mao, A. Ou, C. Schmidt, S. Steffl, J. Wright, I. Stoica, J. Ragan-Kelley, K. Asanovic, B. Nikolic, and Y. S. Shao, "Gemmini: Enabling systematic deep-learning architecture evaluation via full-stack integration," in *Proceedings of the 58th Annual Design Automation Conference (DAC)*, 2021.

[11] N. Neves, P. Tomás, and N. Roma, "Adaptive In-Cache Streaming for Efficient Data Management," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 25, no. 7, pp. 2130–143, 7 2017.