A functional validation framework for the Unlimited Vector Extension

Ana Beatriz Fernandes
Instituto de Telecomunicações
Dept. of Electrical and Computer Eng.
University of Coimbra, Portugal
ana.fernandes@co.it.pt

Nuno Neves
INESC-ID
Instituto Superior Técnico
Universidade de Lisboa, Portugal
nuno.neves@inesc-id.pt

Luís Crespo
INESC-ID
Instituto Superior Técnico
Universidade de Lisboa, Portugal
luis.miguel.crespo@tecnico.ulisboa.pt

Pedro Tomás
INESC-ID
Instituto Superior Técnico
Universidade de Lisboa, Portugal
pedro.tomas@inesc-id.pt

Nuno Roma
INESC-ID
Instituto Superior Técnico
Universidade de Lisboa, Portugal
nuno.roma@inesc-id.pt

Gabriel Falcao
Instituto de Telecomunicações
Dept. of Electrical and Computer Eng.
University of Coimbra, Portugal
gff@co.it.pt

ABSTRACT

The Unlimited Vector Extension (UVE) was already proposed to tackle the limitations of current state-of-the-art Vector-Length Agnostic (VLA) extensions. This is a new Instruction Set Architecture (ISA) extension that aims to reduce loop control and memory access indexation overheads, as well as memory access latency, joining data streaming and Single Instruction, Multiple Data (SIMD) processing. This ISA extension has already been validated in a cycle-accurate simulator, *gem5*, with a first implementation made on an out-of-order processor model, based on the ARM Cortex-A76. However, as compilation support is currently being developed, and several shortcomings and improvements on the existing specification have been identified, an increasing need to efficiently run and validate UVE code has surged. As such, support for UVE has been added to the *Spike* simulator. This is the golden reference functional RISC-V ISA simulator, written in C++. To achieve this, the simulator has been extended to accommodate for the necessary architecture changes, such as new registers that hold the data streams (streaming registers) together with a convenient Streaming Unit that emulates the configuration and manipulation of the streams. The result is a powerful tool that provides the possibility to validate all current features and improvements of UVE, along with some preliminary code obtained from the compiler currently under development.

CCS CONCEPTS

• **Computer systems organization → Single instruction, multiple data**; *Reduced instruction set computing*; **Data flow architectures**; • **Computing methodologies → Simulation tools**.

CAMS'23, October 2023, Toronto, Canada

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

KEYWORDS

ISA SIMD Extensions, Data Streaming, RISC-V, Unlimited Vector Extension, Simulation Tools, Data Flow Architecture

ACM Reference Format:
Ana Beatriz Fernandes, Nuno Neves, Luís Crespo, Pedro Tomás, Nuno Roma, and Gabriel Falcao. 2023. A functional validation framework for the Unlimited Vector Extension. In *Proceedings of ACM Conference (CAMS'23).* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

In the last few decades, there has been an increasing need to improve processors' performance, as computational and memory-intensive applications become more common (e.g. Machine Learning and Sparse Linear Algebra). However, with the end of Dennard Scaling and Moore's Law, more traditional methods of improving performance have been revealed to be insufficient, such as increasing the clock frequency and the use of cache memory.

Several solutions have been proposed and are now widely established, such as Instruction-Level Parallelism (ILP) and Data-Level Parallelism (DLP), common in modern high-performance processors. The latter, hidden in Single Instruction, Multiple Data (SIMD) units [8], allow the simultaneous processing of multiple data elements. To take advantage of this, a plethora of SIMD Instruction Set Architectures (ISAs) has been developed, such as Arm NEON [1] and x86 AVX [11], focused on operating on fixed-size registers. However, because a vector's optimal size depends on the application, this approach presents some limitations. To overcome this, other vector-length agnostic extensions have emerged, particularly the RISC-V Vector Extension (RVV) [24] and the Scalable Vector Extension (SVE) [22], which allow for the size of the vector register to be defined at runtime. This means that different processors, with different application requirements, can adopt distinct vector sizes, with no need to modify the source code. However, a new problem arises with these extensions, as predicate [2] and vector control instructions become necessary to disable elements outside loop bounds, which leads to more loop instructions [17], and thus more overhead and decrease of performance.

The RISC-V Unlimited Vector Extension (UVE), proposed and developed by Domingos et al. [8], joins two promising solutions for improving performance: scalable SIMD extensions and data

streaming. RISC-V was chosen as the base ISA due to its open-source nature, as well as its simple and extensible instruction set. By relying on data streaming, this novel RISC-V ISA extension has several improvements when compared to the ones mentioned above, such as decoupled memory accesses, indexing-free loops, simplified vectorisation, and implicit load/store operations [8]. The streaming paradigm allows for the configuration of memory access patterns at software level and the subsequent data fetching in the background, a clear step towards improving the memory access latency and throughput. This paradigm shift was already demonstrated in a proof-of-concept *gem5* implementation of UVE on an out-of-order processor model, based on the ARM Cortex-A76. It showed that it can improve the performance of a processor by up to 2.4 times when compared to other state-of-the-art implementations [8].

In accordance, this work exploits a prominent research trend that considers the use of unconventional architectures to improve the attained processor's performance. In particular, it presents a new modelling, simulation and validation tool to support the development of UVE, by not only independently validating the existing specification, but also introducing streaming support on *Spike* [18], on which several existing instructions were added, tested and, whenever pertinent, modified. Before this contribution, this process was very time-consuming and tightened to the several constraints imposed by *gem5*. Hence, it now becomes much more efficient with this new tool, as *Spike* offers a simpler instruction implementation pipeline. For this to be possible, the simulator was expanded to include a Streaming Unit (SU), similar to RVV's Vector Unit already present in the simulator. Moreover, a new RISC-V extension was added to the simulator, where many of the existing UVE instructions were added.

In order to test the streaming mechanisms and validate the functional behaviour of the chosen instructions, a subset from the benchmarks that were considered in UVE's proposal [8] was used, mainly based on Polybench/C [1]. It should be noted that for applications it is not yet possible to automatically generate UVE code from regular C code, which means that some benchmarks had to be manually written. This had already been done for previous works, but the available code was revised and compared to code generated from the compiler currently being developed [13] whenever possible.

The developed framework and supporting documentation are publicly available online [2]. This repository contains ongoing work and is thus subject to continual updates.

2 DATA STREAMING AND UVE

Memory access is the most time and energy-consuming operation in modern computer architectures [5], so it is natural that this is the main target of optimisation attempts. While cache structures greatly improve access latencies, they are dependent on data locality, which is not always guaranteed. Moreover, in applications with complex access patterns, it is often not possible to efficiently make use of these structures. Furthermore, if there is a large volume of data to be loaded/stored, particularly in multi-core systems, instances of cache contention (i.e. when multiple cores attempt to update the same cache line) and energy consumption rise [10]. This means that adapting the data communication scheme to the running application is crucial for performance increase. One re-emerging technology aiming to tackle this problem is data streaming [8, 13, 16, 20, 21, 23], which decouples memory accesses from data processing, effectively masking data transfers behind computation [7].

A stream is essentially a predicable vector of data elements that are processed sequentially. Each element of a stream is subject to the same set of operations and is discarded after the computation is complete. These structures rely almost solely on spatial locality, which means that the order in which the data is going to be consumed can be specified beforehand [7]. This is possible through data pattern descriptors, such as those proposed and developed in [8, 14, 15]. Understanding this representation model is pivotal to understanding UVE. Hence, the fundamentals of data streaming and pattern description are described next.

Any regular n-dimensional access sequence can be represented by the following affine function:

$$y(X) = y_{base} + \sum_{k=0}^{dim_y} x_k \times S_k, \tag{1}$$

with $X = x_0, ..., x_{dim_y}$ and $x_k \in [O_k, E_k + O_k]$.

This means that a stream access $y(X)$ is described as the sum of the base address of an n-dimensional variable (y_{base}) with dim_y pairs of indexing variables (x_k) and their respective strides (S_k), each k corresponding to a dimension of the pattern. E_k corresponds to the number of elements in each k dimension and O_k to the indexing offset. Because x_0 has $O_0 = 0$, it is equal to the base address of the variable [8]. Moreover, through a combination of affine functions of this kind, highly complex patterns can be attained, by assigning the base address and/or the offset of a function to the result of another one. Lastly, indirect memory accesses can also be described by taking the data obtained by the addresses generated by an affine function and injecting them into the aforementioned variables of another function.

The proposed pattern representation model results from the encoding of the variables associated with each pattern dimension of the function described in Equation (1). This representation is based

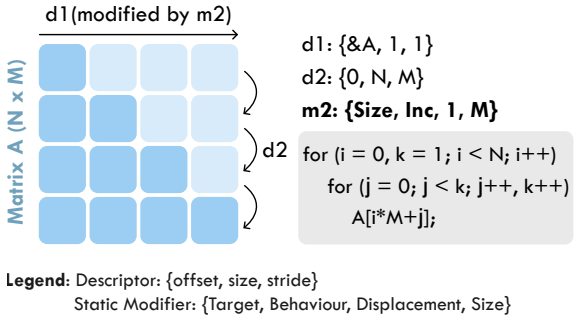

Figure 1: Triangular access pattern description, where a static modifier is applied to increment the size of the first dimension.

[1] https://web.cs.ucla.edu/ pouchet/software/polybench/
[2] https://github.com/hpc-ulisboa/UVE2

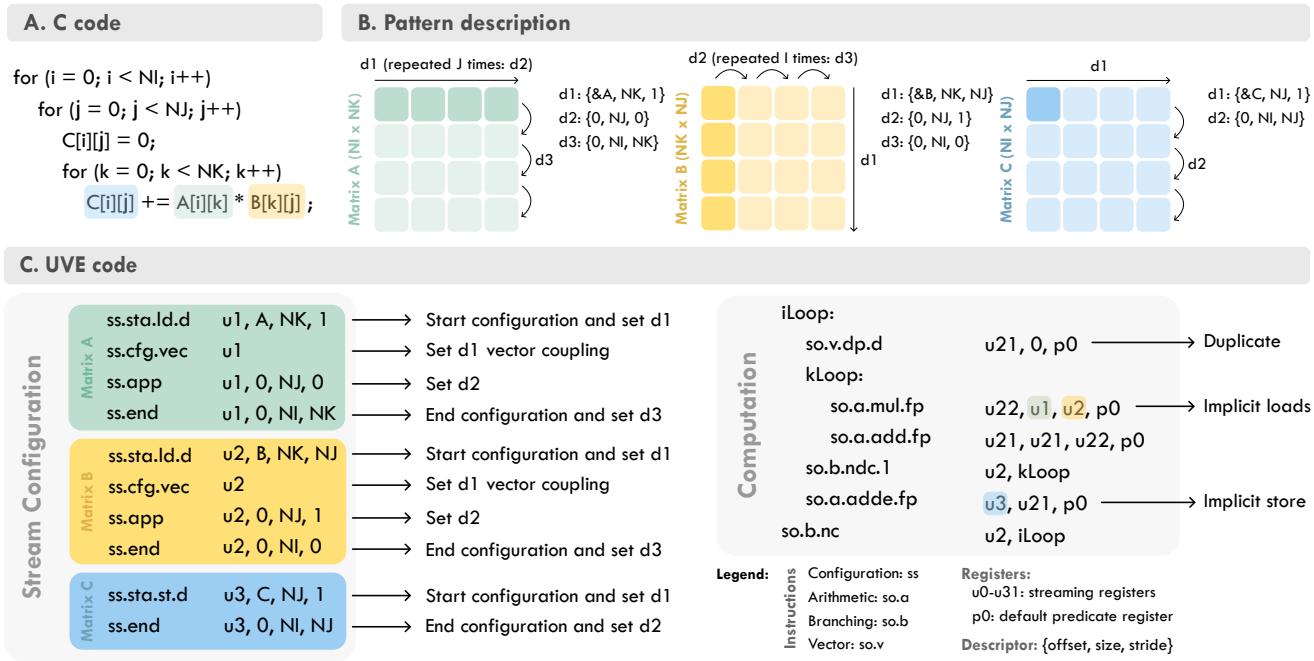

Figure 2: Exemplification of two-matrix multiplication from (A) the C source code, through (B) the pattern description of each matrix data access and (C) the resulting UVE configuration and computation kernels.

on descriptors and modifiers (see Figure 1), defined in a set of dedicated instructions in UVE. Simple descriptors, that remain constant throughout execution, are exemplified in Figure 2, which is part of the kernel used in the *trisolv* benchmark (see Section 4). There are two types of optional modifiers, which when associated with a certain dimension of the descriptor are able to alter its parameters, allowing the modelling of inter-loop control dependencies that arise when loop conditions are affected by an outer loop. On the one hand, *static modifiers* are able to add or subtract a certain displacement to any of the dimension's parameters. On the other hand, *indirect modifiers* allow for the substitution of these parameters with pointers to data obtained from another stream. This makes it possible to create complex pattern descriptors, which are common in a plethora of applications, such as Sparse Algebra and Data Mining.

UVE adds 32 vector registers to the base ISA (named from *"u0"* to *"u31"*). The length of each vector is unlimited, but a minimum value is defined, equal to the width of the supported data types (*byte*, *half-word*, *word*, and *double-word*), therefore set between 8 and 64 *bytes*, restricted to powers of two. Each of these vectors 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*). Register *p0* is hardwired to 1, 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 are used in the configuration of the other eight.

There are currently 60 major instructions, out of which 26 correspond to integer operations, 15 to floating-point operations and 19 are related to memory manipulation, totalling about 450 instructions when considering the variations of each one.

Furthermore, UVE not only lets one describe data streams through the ISA, but it also defines the operation of the supporting microarchitecture to manipulate them, consisting in a dedicated *Streaming Engine*, along with other minor structures that extend the processor in order to fully support this ISA extension.

Because *Spike* is a functional simulator based on a somewhat high level of abstraction from the real hardware, the added structures do not fully mimic the proposed microarchitecture, namely the memory hierarchy, pipelining and Load/Store FIFOs, but are implemented to respect the instruction set extension specification.

3 UVE VALIDATION FRAMEWORK

Having proven its great potential [8], UVE will benefit from an efficient tool to validate every aspect of its specification, so that it can be further improved and expanded to support new and more complex applications. The developed framework is hereby described in detail and its structure is represented in Figure 3.

3.1 Simulator

As noted by Roelke and Stan [19], there is usually a compromise between simulation accuracy and speed when choosing between the various RISC-V simulators available. As such, *Spike* was chosen as the most appropriate tool to continue this development. Although it does not allow cycle-accurate precision, it is the golden reference

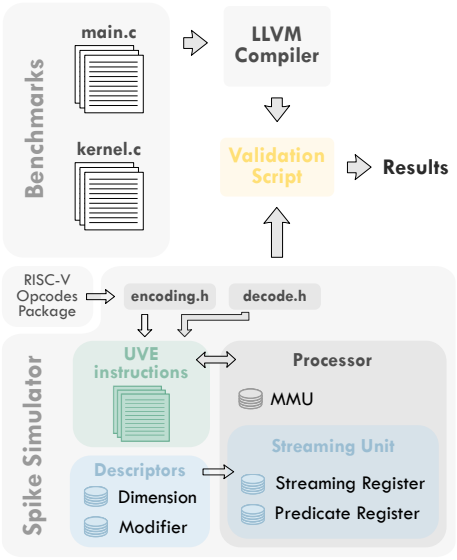

Figure 3: Framework structure.

functional RISC-V ISA software simulator and is widely used as the proof-of-concept target for every RISC-V extension [6, 12]. In fact, despite QEMU appearing to be slightly more accurate [19], it is a much bigger and more complex project, as it targets multiple architectures, not only RISC-V, and is thus more difficult to modify, something that is necessary in order to create UVE support. This is pointed out by Henriques [9], who already used the *Spike* simulator to implement some UVE instructions and whose work laid the foundation for the development of the currently proposed validation framework.

Spike is currently at Version 1.1.0 and already supports many RISC-V ISA features, along which is the RISC-V Vector Extension, which served as a base for the developed Streaming Unit (SU) – the UVE's equivalent to its Vector Unit. However, upon analysing the implementation of several extensions on the simulator, it became clear that UVE's implementation structure would be very different. This is mainly due to the way the simulator's source code is written, heavily dependent on macros defined in multiple files and with little to no documentation. This resulted in code structured in a very different way than the rest of the simulator and its supported extensions, albeit more comprehensible.

3.1.1 Streaming simulation infrastructure. The focal component of the developed simulator is the SU, a new class that has access to the streaming and predicate registers. This unit mimics some parts of the proposed *Streaming Engine* [8], specifically the *Stream Table* and the *Stream Processing Module*, as well as the remaining infrastructure responsible for the memory accesses (see Figure 4). Each register may or may not be associated to a stream, and this module is responsible for the implicit loading and storing of data, as well as the iteration of the streams (by the *Address Generator*). For the desired functional evaluation, the *Load/Store FIFOs* and the *Stream Scheduler*, represented in Figure 4, were not needed, as streams are iterated as they are being consumed, with each computation instruction triggering the iteration of the source streams (implicit loading) and the destination streams (implicit storing). The resulting elements are immediately placed in the associated registers and the *End Of Dimension* flags are updated and saved to the *Stream Table*. The iteration and address generation parts work very similarly to the proposed configuration and are implemented in a different class, *Dimension*, which has access to the *Modifier* class, where static modifiers are implemented. Each streaming register, when associated to a stream, is therefore also associated to n dimensions and respective modifiers, if such is the case.

Furthermore, predication support was developed at the instruction level, which means that the predicate values never reach the SU, for simplicity. A predicate register has a fixed vector size of 64 *bytes*, and a predicate is thus evaluated according to the datatype of the instruction's 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 [8].

3.1.2 Modified files. Several source files were modified to add the necessary structures to support UVE (e.g. decoding functions for each instruction argument), according to the ISA encoding. These functions, divided into different types of instructions, followed the same pattern as already existing ones, some even being direct copies, so that there is complete flexibility in case the UVE encoding is changed. In that case, it is not necessary to alter each instruction if, for example, one of the source registers is differently encoded. It is only required that the decoding function corresponding to its type is updated accordingly.

For the simulator to recognise the new instructions, the file that holds all the ISA encoding, `encoding.h`, must 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 and UVE's predicate registers, and immediate encoding was added to the file `constants.py`.

Lastly, the new extension was added to file `riscv/riscv.mk.in`, identically to what is done to the native ones, so each new instruction was included in the variable `riscv_insn_ext_uve`. In this file every new source and header file was also added to variables `riscv_srcs` and `riscv_install_hdrs`, respectively, so that they could be recognised during the compilation of the simulator.

3.1.3 New files. The various new classes priorly mentioned are defined in files `descriptors.h` (dimensions and modifiers) and `streaming_unit.h` (registers and SU).

Furthermore, each instruction has a corresponding *header* file in the `riscv/insns` folder. While compiling the simulator, these files will be used to create copies of the `riscv/insn_template.cc` file for each instruction, responsible for the generation of the various versions of the instruction (e.g. 32/64 bit). The obvious implication is that the developed code for an instruction exists inside an external function, therefore header file inclusion is not allowed and only some variables are accessible, namely the processor, the executed instruction and the process counter. It is through the processor that each instruction can access the Main Memory Unit (MMU), as well as the SU and its registers. The executed instruction, an `insn_t`

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

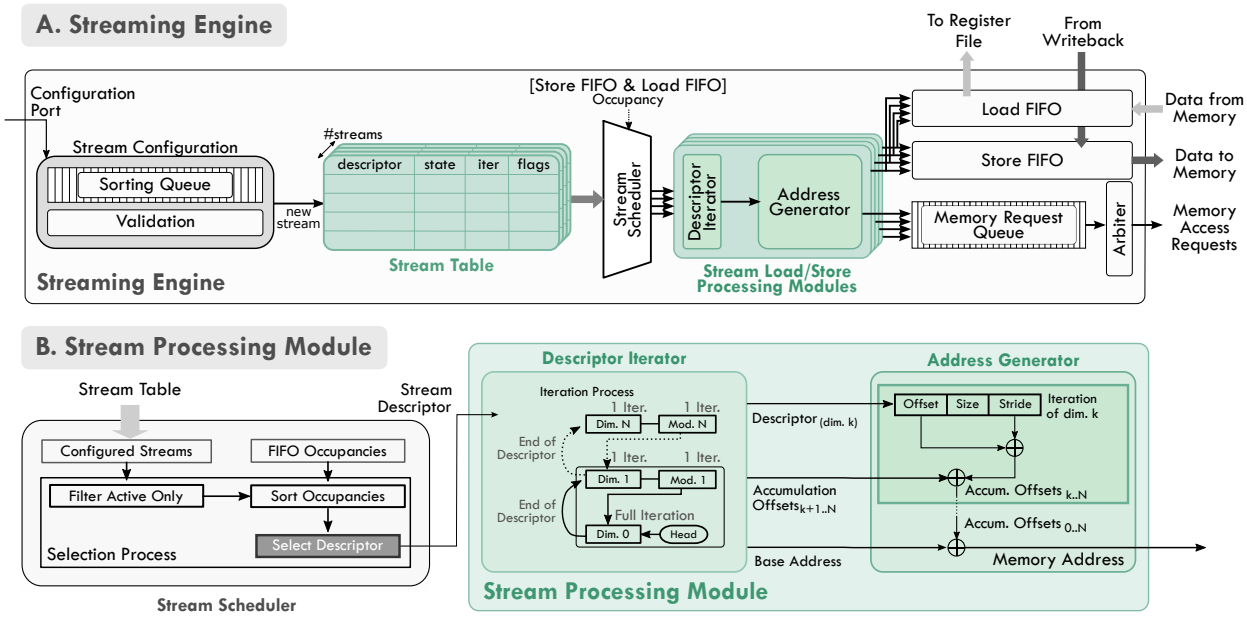

Figure 4: (A) Streaming Engine and (B) Stream Processor Module proposed by Domingos et al. [8], now emulated on *Spike*.

object, has access to the opcode decoding functions, allowing the instruction code to access its operands. The process counter is mainly used in branching instructions.

4 EXPERIMENTAL RESULTS

4.1 Methodology

In order to validate the UVE ISA functional simulation, various benchmarks from a wide range of application domains, such as memory access, linear algebra/BLAS and stencil, were chosen. These benchmarks were either hand-coded in order to have its corresponding UVE implementation, inserted in the original source code using *inline assembly*, or even generated by an adaptded version of the LLVM compiler that, while currently unavailable to the public, is undergoing preliminary testing.

Figure 2.C shows how a simple matrix multiplication ($C = A \times B$) can be coded with UVE. In this example, *u1* and *u2* are streaming registers configured with load data streams from matrices A and B and *u3* is associated to the store stream, corresponding to matrix C, which holds the computation results. This example is also part of the kernel used in benchmark *3mm*, already implemented and tested on *Spike* (see Section 4.2 and Figure 5).

4.2 Evaluation

The described framework is currently able of simulating most of the UVE specification, which means that data streaming capability has been successfully added to *Spike*, as well as many instructions from the proposed ISA. It currently supports multi-dimensional pattern descriptors, as well as static modifiers, although indirection is not yet implemented. Stream-based branching and predication are also supported, as well as multiple arithmetic and vector operations on the streaming registers. In total, more than 100 instructions have been implemented and validated on *Spike*, which can be categorised as follows:

- Arithmetic (41)
- Branching (16)
- Predication (15)
- Vector (8)
- Stream Configuration (21)

With these fully functional instructions, several benchmarks can already be ran on the simulator, as summarised in Figure 5. All these benchmarks, which had been previously used for validation of the UVE ISA and supporting microarchitecture on *gem5*, outputted the same expected behaviour in *Spike*, proving the correct functioning of the developed Streaming Unit and the added instructions on the this simulator.

5 RELATED WORK

The main focus of the presented work is the development of a new simulation environment for UVE, where the major difference from previous works is the chosen base tool. In this section, an overview of the legacy *gem5* UVE simulation framework is presented.

Benchmarks	# Streams	# Kernels *	Max. Loop Nesting	Memory Access Pattern
A. Memcpy (MEMORY)	1	1	1	1D
B. SAXPY (BLAS)	3	1	1	1D
C. 3mm (ALGEBRA)	3	3	3	3D
D. Trisolv (ALGEBRA)	5	1	2	2D +Static Modifier
E. Jacobi-1D (STENCIL)	8	2	1	1D
D. Jacobi-2D (STENCIL)	12	2	2	2D

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

Figure 5: Benchmarks used for testing and respective characteristics.

gem5 was notoriously used in UVE's proposal by Domingos et al. [8], to provide a reliable performance evaluation with cycle-accurate precision based on the adaption of an out-of-order processor model architecture, as it is extendable to support custom ISAs, as well as microarchitecture models [3, 4, 8, 19].

Despite being an open-source project, *gem5* has little documentation available, similarly to *Spike*. However, it is a much more complex tool, which resulted in a long and difficult process of modifications to support UVE. The source code of the simulator was extensively changed, to support new vectorial and predicate registers, where scalability (e.g. variable vector register length, with the element width as a part of the register) was a major obstacle to overcome. This is due to the simulator not being prepared for this at the ISA level, as well as requiring that the configuration of the architecture is changed at each execution. Furthermore, the instruction set parser was modified to support the new vector registers, as well as the width and valid index register information [8]. Lastly, UVE instructions were added, described in a Domain-Specific Language (DSL) based on C++ and Python, which, on the one hand, allowed templating for multiple instructions and code reuse, but on the other hand required many different templates to be developed, such as for instructions with different operands, which are extremely common.

Because this simulator relied on the implementation of the supporting microarchitecture for the validation of UVE, as it depends on a *Streaming Engine*, the instruction set had not been simulated independently until now. The proposed *Spike*-based framework made it possible to focus solely on the instruction's behaviour, detaching the ISA development from implementation details prone to specification errors.

Lastly, simulation platforms alternative to *Spike* exist, such as QEMU[4] and Chisel[5]. While the former is closely related to *Spike*, it is a more complex project, and therefore lacks the simplicity required for an efficient and continuously changing framework. The latter could be used to create an RTL simulation. While this was not the goal of this work, it could be useful, as the proposed framework exists within a bigger project on UVE currently under development, where this type of validation is appropriate.

[4]https://www.qemu.org/
[5]https://www.chisel-lang.org/

6 CONCLUSION

In this paper, a new validation framework for the UVE ISA extension based on the *Spike* RISC-V simulator is presented. This new simulation tool provides efficient development means and functional evaluation for this ISA extension and of its supporting microarchitecture. A Streaming Unit, responsible for the management of the streams, and most of the existing UVE instructions were added to *Spike*, also including support and ensuring validation for the main functionalities already offered by UVE, such as data streaming with implicit loads/stores, predication, and *n*-dimensional pattern description with static modifiers. A representative set of benchmarks was tested and verified, confirming the previous results that had been obtained with a *gem5* legacy UVE simulator. Furthermore, this tool was also used to validate the preliminary results from the UVE-LLVM compiler that is currently under development, and has proven to be useful both in the development of UVE applications and in the development of the UVE extension itself.

6.1 Future Work

Although the most part of the instruction set has already been added to the framework, some instructions of the existing ISA are yet to be implemented, namely logical and stream configuration instruction. The latter have to be accompanied by the implementation of indirection, which is not yet supported.

In addition, the *Spike* simulator's disassembler still has no information about the added extension, which makes the use of the *debugger* less straightforward. This can be improved in the future, making the developed tool much more useful by improving code review and correction.

Lastly, the LLVM compiler toolchain currently under development is to be tested on this framework. Once it is released, it will integrate this project, removing the need to hand-code applications where UVE is to be used. This will allow anyone to develop software taking advantage of this new ISA extension.

ACKNOWLEDGMENTS

This work was supported by national funds through Fundação para a Ciência e a Tecnologia (FCT), under projects UIDB/50008/2020, UIDB/50021/2020, EXPL/EEI-HAC/1511/2021, 2022.06780.PTDC, 2022.11626.BD, and from the European High Performance Computing Joint Undertaking (JU) under Framework Partnership Agreement No 800928 and Specific Grant Agreement No 101036168 (EPI SGA2). The JU receives support from the European Union's Horizon 2020 research and innovation programme and from Croatia, France, Germany, Greece, Italy, Netherlands, Portugal, Spain, Sweden, and Switzerland.

REFERENCES

[1] Arm. 2011. *Introducing NEON Development Article.* https://developer.arm.com/documentation/dht0002/a/

[2] Adrian Barredo, Juan M. Cebrian, Miquel Moreto, Marc Casas, and Mateo Valero. 2020. Improving Predication Efficiency through Compaction/Restoration of SIMD Instructions. In *2020 IEEE International Symposium on High Performance Computer Architecture (HPCA).* IEEE, San Diego, CA, USA, 717–728. https://doi.org/10.1109/HPCA47549.2020.00064

[3] Nathan Binkert, Bradford Beckmann, Gabriel Black, Steven K. Reinhardt, Ali Saidi, Arkaprava Basu, Joel Hestness, Derek R. Hower, Tushar Krishna, Somayeh Sardashti, Rathijit Sen, Korey Sewell, Muhammad Shoaib, Nilay Vaish, Mark D. Hill, and David A. Wood. 2011. The Gem5 Simulator. *SIGARCH Comput. Archit. News* 39, 2 (aug 2011), 1–7. https://doi.org/10.1145/2024716.2024718

[4] N.L. Binkert, R.G. Dreslinski, L.R. Hsu, K.T. Lim, A.G. Saidi, and S.K. Reinhardt. 2006. The M5 Simulator: Modeling Networked Systems. *IEEE Micro* 26, 4 (2006), 52–60. https://doi.org/10.1109/MM.2006.82

[5] Amirali Boroumand, Saugata Ghose, Youngsok Kim, Rachata Ausavarungnirun, Eric Shiu, Rahul Thakur, Daehyun Kim, Aki Kuusela, Allan Knies, Parthasarathy Ranganathan, and Onur Mutlu. 2018. Google Workloads for Consumer Devices: Mitigating Data Movement Bottlenecks. In *Proceedings of the Twenty-Third International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS '18).* Association for Computing Machinery, New York, NY, USA, 316–331. https://doi.org/10.1145/3173162.3173177

[6] Chipyard. 2019. *The RISC-V ISA Simulator (Spike) - Chipyard 1.8.1 documentation.* https://chipyard.readthedocs.io/en/latest/Software/Spike.html

[7] Silviu Ciricescu, Ray Essick, Brian Lucas, Phil May, Kent Moat, Jim Norris, Michael Schuette, and Ali Saidi. 2003. The Reconfigurable Streaming Vector Processor (RSVPTM). (2003).

[8] Joao Mário Domingos, Nuno Neves, Nuno Roma, and Pedro Tomás. 2021. Unlimited Vector Extension with Data Streaming Support. In *2021 ACM/IEEE 48th Annual International Symposium on Computer Architecture (ISCA).* IEEE, Valencia, Spain, 209–222. https://doi.org/10.1109/ISCA52012.2021.00025

[9] Luís Henriques. 2022. *Automatic Streaming for RISC-V via Source-to-Source Compilation.* Master's thesis. Universidade do Porto, Porto. https://hdl.handle.net/10216/142750

[10] Rakesh Kumar, Timothy G. Mattson, Gilles Pokam, and Rob Van Der Wijngaart. 2011. *The Case for Message Passing on Many-Core Chips.* Springer New York, New York, NY, 115–123. https://doi.org/10.1007/978-1-4419-6460-1_5

[11] Chris Lomont. 2009. *Introduction to Intel® Advanced Vector Extensions.* www.obpm.org/download/Intro_to_Intel_AVX.pdf

[12] Christoph Müllner. 2021. *Emulators and Simulators - RISC-V International.* https://wiki.riscv.org/display/HOME/Emulators+and+Simulators#EmulatorsandSimulators-Spike/riscv-isa-sim

[13] Nuno Neves, João Mário Domingos, Nuno Roma, Pedro Tomás, and Gabriel Falcao. 2022. Compiling for Vector Extensions With Stream-Based Specialization. *IEEE Micro* 42, 5 (9 2022), 49–58. https://doi.org/10.1109/MM.2022.3173405

[14] Nuno Neves, Pedro Tomás, and Nuno Roma. 2015. Efficient data-stream management for shared-memory many-core systems. In *2015 25th International Conference on Field Programmable Logic and Applications (FPL).* 1–8. https://doi.org/10.1109/FPL.2015.7293960

[15] Nuno Neves, Pedro Tomás, and Nuno Roma. 2017. Adaptive In-Cache Streaming for Efficient Data Management. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 25, 7 (7 2017), 2130–143. https://doi.org/10.1109/TVLSI.2017.2671405

[16] Tony Nowatzki, Vinay Gangadhar, Newsha Ardalani, and Karthikeyan Sankaralingam. 2017. Stream-Dataflow Acceleration. *SIGARCH Comput. Archit. News* 45, 2 (jun 2017), 416–429. https://doi.org/10.1145/3140659.3080255

[17] Angela Pohl, Mirko Greese, Biagio Cosenza, and Ben Juurlink. 2019. A Performance Analysis of Vector Length Agnostic Code. In *2019 International Conference on High Performance Computing & Simulation (HPCS).* IEEE, Dublin, Ireland, 159–164. https://doi.org/10.1109/HPCS48598.2019.9188238

[18] RISC-V. 2021. *Spike RISC-V ISA Simulator.* https://github.com/riscv-software-src/riscv-isa-sim

[19] Alec Roelke and Mircea R Stan. 2017. RISC5: Implementing the RISC-V ISA in gem5.

[20] Paul Scheffler, Florian Zaruba, Fabian Schuiki, Torsten Hoefler, and Luca Benini. 2021. Indirection Stream Semantic Register Architecture for Efficient Sparse-Dense Linear Algebra. In *2021 Design, Automation & Test in Europe Conference & Exhibition (DATE).* 1787–1792. https://doi.org/10.23919/DATE51398.2021.9474230

[21] Fabian Schuiki, Florian Zaruba, Torsten Hoefler, and Luca Benini. 2021. Stream Semantic Registers: A Lightweight RISC-V ISA Extension Achieving Full Compute Utilization in Single-Issue Cores. *IEEE Trans. Comput.* 70, 2 (2021), 212–227. https://doi.org/10.1109/TC.2020.2987314

[22] Nigel Stephens, Stuart Biles, Matthias Boettcher, Jacob Eapen, Mbou Eyole, Giacomo Gabrielli, Matt Horsnell, Grigorios Magklis, Alejandro Martinez, Nathanael Premillieu, Alastair Reid, Alejandro Rico, and Paul Walker. 2017. The ARM Scalable Vector Extension. *IEEE Micro* 37, 2 (3 2017), 26–39. https://doi.org/10.1109/MM.2017.35 arXiv:1803.06185 [cs].

[23] Zhengrong Wang and Tony Nowatzki. 2019. Stream-Based Memory Access Specialization for General Purpose Processors. In *Proceedings of the 46th International Symposium on Computer Architecture* (Phoenix, Arizona) *(ISCA '19).* Association for Computing Machinery, New York, NY, USA, 736–749. https://doi.org/10.1145/3307650.3322229

[24] A. Waterman and K. Asanovic. 2021. *RISC-V "V" Vector Extension.* https://github.com/riscv/riscv-v-spec/blob/master/v-spec.adoc