This document is a preprint of: V. Razilov, E. Matus and G. Fettweis, "Communications Signal Processing Using RISC-V Vector Extension," in Proceedings of International Wireless Communications and Mobile Computing Conference (IWCMC 2022), Dubrovnik, Croatia, (pp. 690-695), May 2022. DOI:10.1109/IWCMC55113.2022.9824961

Communications Signal Processing Using RISC-V Vector Extension

Viktor Razilov*, Emil Matúš*, and Gerhard Fettweis*†
*Vodafone Chair Mobile Communications Systems, Technische Universität Dresden, Germany
†Barkhausen Institut, Dresden, Germany
{viktor.razilov, emil.matus, gerhard.fettweis}@tu-dresden.de

***Abstract*—Flexible and scalable solutions will be needed for future communications processing systems. RISC-V processors enhanced with vector processing capabilities as specified by the soon-to-be ratified RISC-V vector extension (RVV) pose an interesting base for such systems. Vector processors provide an efficient means of exploiting data-level parallelism, which is heavily present in communications kernels. Furthermore, RVV code is by its design agnostic from the underlying hardware platform which enables scalability. On the exemplary basis of a generalized frequency division multiplexing (GFDM) implementation on a RVV processor, we investigate its baseband processing capabilities and guide through RVV's key features and peculiarities. Our vectorization achieves a speedup of up to 60 times compared to the scalar base case and a throughput of 784 symbols per second. The utilization of 77 % is slightly below more specialized solutions. Nevertheless, this work serves as a baseline for further investigations on flexible and scalable RISC-V vector communications processors.**

***Index Terms*—Communications processing, GFDM, RISC-V, Vector processor.**

I. Introduction

Wireless communication is used in an increasingly diverse set of applications. As the usage scenarios vary, so do the choices of algorithms and prioritized performance indicators [1]. It is anticipated that applications of sixth generation (6G) cellular networks require different minimum latencies and throughputs whose ranges span 3 or 7 orders of magnitude, respectively [2]. This diversity puts a higher stress on the flexibility and scalability of the digital baseband processing hardware.

One technique to accommodate flexibility and performance is the design of application-specific instruction set processors (ASIPs) [3]. However, not every application has a market size that justifies custom chip development and production. Here, versatile and low-cost programmable processors that achieve high performance for communications processing are desirable. Being open, simple, extendable, and supported by popular open-source compiler toolchains GCC and LLVM, RISC-V [4] is a promising candidate as the base instruction set architecture (ISA) of such a system.

There is a working draft of an official RISC-V vector extension (RVV) [5]. Like many single instruction, multiple data (SIMD) extensions, it offers a way of efficiently exploiting data-level parallelism (DLP) [6]. In contrast to packed-SIMD extensions which have a fixed vector length, RVV is vector length agnostic (VLA), making it highly scalable. Idiomatic RVV code runs on any RVV implementation without recoding, from embedded microcontrollers with short vectors to high-performance processors with long vectors.

Because communications signal processing tasks often exhibit considerable DLP, we investigate the feasibility of RVV for flexible, scalable and efficient execution of such number crunching tasks. As an example algorithm, we take generalized frequency division multiplexing (GFDM) [7]. There exist thorough studies on implementation aspects of GFDM [8], [9] and implementations on various processing platforms [8]–[11].

As our contribution, we implement GFDM on RVV and profile the execution on an open-source RVV processor. We point out RVV's peculiarities and performance optimizations for baseband kernels, such as data memory arrangement and vector instruction ordering. We assess the impact of these optimizations and RVV's throughput and utilization in comparison to other GFDM solutions.

The rest of the paper is structured as follows: Section II sheds a light on the background and related work on GFDM and RVV processors. Afterwards, we describe the steps taken to implement and optimize GFDM on RVV in Section III. In Section IV, we evaluate the throughput and compare it to other GFDM solutions. Section V concludes and makes an outlook to further research.

II. Background and Related Work

GFDM is a generalization of orthogonal frequency division multiplexing (OFDM) and considered a candidate waveform for next generation mobile communication. Compared to OFDM, it has higher flexibility, reduced out-of-band emission, and peak-to-average power ratio but also a higher computational complexity [7], [12]. One low-complexity GFDM approach is time-domain processing [13], [14], which is the focus of this paper.

In a GFDM frame, $N = MK$ data symbols are divided into M subsymbols and K subcarriers with data symbol $d_k[m] \in \mathbb{C}$ being placed on subsymbol $m = 0, 1, \ldots, M-1$ of the subcarrier $k = 0, 1, \ldots, K-1$. Each subcarrier is modulated with a circularly frequency and time shifted version of the prototype pulse shaping filter g. Thus, one obtains, for $n = lK + i$, $l = 0, 1, \ldots, M-1$, $i = 0, 1, \ldots, K-1$, the GFDM

This document is a preprint of: V. Razilov, E. Matus and G. Fettweis, "Communications Signal Processing Using RISC-V Vector Extension," in Proceedings of International Wireless Communications and Mobile Computing Conference (IWCMC 2022), Dubrovnik, Croatia, (pp. 690-695), May 2022. DOI:10.1109/IWCMC55113.2022.9824961

signal [8]

$$x[n] = \sum_{k=0}^{K-1} \sum_{m=0}^{M-1} d_k[m] g[(n - mK) \bmod N] e^{j2\pi \frac{kn}{K}} = \sum_{m=0}^{M-1} g[(n - mK) \bmod N] \underbrace{\sum_{k=0}^{K-1} d_k[m] e^{j2\pi \frac{kn}{K}}}_{D_n[m] = \mathcal{F}^{-1}\{d_k[m]\}} . \quad (1)$$

Hence, time-domain GFDM resolves to an inverse discrete Fourier transform (IDFT) followed by filtering of the transformed data. The latter consists of 3 nested loops of complex multiply-accumulation (MAC). As IDFTs have been the subject of numerous investigations already, we focus on the filtering part, in line with [8], [10].

The work in [8], [9] provide an in-depth analysis of the filter computation and its numerical precision requirements, loop ordering, and vectorization methodology. Based on the insights, they implement the filter on a vector digital signal processor (DSP) [8]. An ASIP approach that is optimized for low energy consumption is presented by [10] and a high-performance field-programmable gate aray (FPGA) approach by [11].

To the authors' knowledge, there is no RISC-V-based GFDM solution in literature. There are, however, wireless-communications-targeting RISC-V ISA extensions like complex arithmetic instructions with minimal power overhead [15] or custom instructions for forward error correction codes [16]. ARM-based platforms have also been used for communications processing, for example, in conjunction with embedded FPGAs [17].

ASIPs enhance performance by fusing arithmetic and logic operations into a single instruction that would otherwise cost several basic instructions. Another way of minimizing instruction overhead is SIMD computation, where a single instruction operates on vectors of data. Currently, two RISC-V SIMD extensions have gained traction: The packed-SIMD "P" extension [18] and RVV [5]. Packed-SIMD instructions explicitly encode the vector length and the data width resulting in an explosion of the instruction set size and low portability. RVV on the other hand decouples software from the underlying hardware by means of a special instruction to dynamically set the vector length. ARM's scalable vector extension (SVE) [19] uses loop predication, to similar effect, with an associated overhead of around 10 % [20]. In addition, vector processors tend to decouple the vector length from the number of parallel computational units. The vector length is determined only by the size of the vector register file (VRF). Both kinds of decoupling allow for tradeoffs between area, power, and throughput such that the hardware can be tuned to the needs of the application without having to recode the software.

Even though the RVV ratification is as of the time of writing still a work-in-progress, several RVV platforms for various use cases have already been published: E.g., Ara [21] and RISC-V^2 [22] for high-performance computing, [23] for embedded microcontrollers, and Vicuna [24] for real-time computing.

```
1  input: IDFT result D ∈ ℂ^N,
2       duplicated filter coefficients ḡ ∈ ℂ^{2N}
3  output: GFDM signal x ∈ ℂ^N
4
5  for l ← 0 to M − 1 do
6    for i ← 0 to K − 1 do
7      a ← 0;
8      for m ← 0 to M − 1 do
9        a ← a + D[mK + i] ḡ[(M − 1 + m − l)K + i];
10     end
11     x[lK + i] ← a;
12   end
13 end
```

Fig. 1. GFDM scalar variant. g is duplicated in memory to avoid modulo operations.

```
1  input: IDFT result D ∈ ℂ^N,
2       duplicated filter coefficients ḡ ∈ ℂ^{2N}
3  output: GFDM signal x ∈ ℂ^N
4
5  γ ← 0;
6  ξ ← 0;
7  for l ← 0 to M − 1 do
8    δ ← 0;
9    γ ← γ + (M − 1 − l)K;
10   k ← K;
11   while k > 0 do
12     i ← min(k, n_v);
13     a⃗ ← 0⃗;
14     for m ← 0 to M − 1 do
15       a⃗ ← a⃗ + D[δ ... δ + i − 1] ∘ ḡ[γ ... γ + i − 1];
16       δ ← δ + K;
17       γ ← γ + K;
18     end
19     x[ξ ... ξ + i − 1] ← a⃗;
20     ξ ← ξ + i;
21     δ ← δ − N + i;
22     γ ← γ − N + i;
23     k ← k − i;
24   end
25 end
```

Fig. 2. GFDM vector variant. The vector element count n_v is given by the vector length of the platform VLEN divided by the data width w_d. $D[a \dots b]$ denotes the subvector with the elements a to b of vector D, and $\circ$ the element-wise product of two vectors.

As a scalable high-throughput example platform, we target Ara for our investigation.

III. Implementation

A. Vectorization

One scalar variant of a GFDM modulation is given in Fig. 1. To avoid modulo operations when accessing the filter coefficients g, they are duplicated in memory by concatenation $\bar{g} = g.g$ [10]. Instructions related to the MAC and the two loads in the innermost loop will be executed M^2K and those related to storing the result MK times.

With vector processing, these repetitions can be greatly reduced. While there are multiple ways to vectorize the

This document is a preprint of: V. Razilov, E. Matus and G. Fettweis, "Communications Signal Processing Using RISC-V Vector Extension," in Proceedings of International Wireless Communications and Mobile Computing Conference (IWCMC 2022), Dubrovnik, Croatia, (pp. 690-695), May 2022. DOI:10.1109/IWCMC55113.2022.9824961

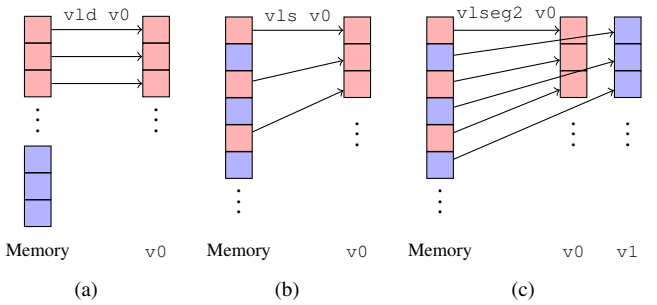

Fig. 3. Options for data memory arrangement and vector loading. Real parts are colored red, imaginary blue. (a) Separated memory arrangement and unit-stride load. (b) Interleaved memory arrangement and strided load. (c) Interleaved memory arrangement and unit-stride segment load.

Listing 1
SIMPLIFIED C CODE FOR THE INNERMOST LOOP IN FIG. 2.

```
// Input: int16_t *Dr, *Di, *gr, *gi;
//        int N, K;
for (int m = 0; m < M; m++) {
  vld(vDr, Dr);            // I
  vld(vDi, Di);            // II
  vld(vgr, gr);            // III
  vld(vgi, gi);            // IV
  vwmacc(var, vDr, vgr); // V
  vrsub(vDi, vDi, 0);      // VI
  vwmacc(var, vDi, vgi); // VII
  vrsub(vDi, vDi, 0);      // VIII
  vwmacc(vai, vDr, vgi); // IX
  vwmacc(vai, vDi, vgr); // X
  Dr += K; Di += K; gr += K; gi += K;
}
```

algorithm with different loop orders and vector loop selections, [8] demonstrated the variant exemplified in Fig. 2 to yield the highest throughput. It is worthy to note that the algorithm in Fig. 2 does not assume a specific vector length. In VLA fashion, it adapts its behavior to any vector element count n_v that is given by the hardware at runtime. $n_\mathrm{v} = \frac{\mathrm{VLEN}}{w_\mathrm{d}}$ depends on the hardware vector length VLEN and the data width w_d.

After vectorization, vector load and vector MAC operations are issued $\left\lceil \frac{M^2K}{n_\mathrm{v}} \right\rceil$ and vector store operations $\left\lceil \frac{MK}{n_\mathrm{v}} \right\rceil$ times. Control flow instructions related to the middle loop in Fig. 2 occur only $\left\lceil \frac{K}{n_\mathrm{v}} \right\rceil$ times. The instruction bandwidth is thus reduced, especially when n_v is high.

B. Number representation

A number representation with finite number of bits needs to be selected in digital systems. Unlike the scalar RISC-V, where numbers have a minimum precision of 32 bit, RVV supports 8-, 16-, 32-, and 64-bit fixed-point and floating-point numbers enabling a tradeoff between precision and throughput. From the available options, fixed point numbers with $w_\mathrm{d} = 16\,\mathrm{bit}$ precision for D, g, and x and $w_\mathrm{a} = 32\,\mathrm{bit}$ for the accumulator vector $\vec{a}$ for each of the real and the imaginary parts are of sufficient precision [8]. Before the result x is written to memory, it is shifted and converted to the format of the input data. Its width is accordingly $w_\mathrm{x} = w_\mathrm{d} = 16\,\mathrm{bit}$.

C. Data Memory Arrangement

Vector processors usually offer many powerful ways of vector loading to serve a diverse set of data arrangements. There are two usual ways of arranging real and imaginary parts of complex-number data in memory: separated and interleaved (c.f. Fig. 3). For real-number vector computing, as in RVV, we need to hold each part in a different vector register. The combinations of memory arrangement and corresponding vector loading in Fig. 3 ensure this condition.

Separated arrangement, as in Fig. 3a, allows unit-stride loads. For interleaved arrangement, one could issue loads with a stride of two (c.f. Fig. 3b). But this may underutilize the memory interface which usually serves contiguous memory segments. A more suitable alternative is the unit-stride segment load depicted in Fig. 3c, where the consecutive elements are placed into multiple vector registers in an alternating fashion. This load utilizes the full memory interface and reduces the number of instructions as both, the real and the complex part, are loaded with a single instruction.

To the authors' knowledge, there is however no RVV processor supporting segmented load available yet. Therefore, we only investigate the first two options.

D. Partitioning the VRF

RVV prescribes 32 named vector registers, each containing VLEN bits. However, we only need 4 of normal length for the real and imaginary parts of D and g and 2 of double length for $\vec{a}$ leading to 8 normalized vector registers. We avoid the resulting VRF underutilization with the register grouping feature offered by RVV: LMUL consecutive vector registers are treated as a single vector with the effective element count $\hat{n}_v = \mathrm{LMUL} \times n_\mathrm{v}$. A length multiplier of $\mathrm{LMUL} = 4$ makes use of the entire VRF.

E. Complex MAC with RVV instructions

Another porting challenge is the lack of complex-number arithmetic in RVV. The complex MAC (CMAC) can be emulated with the provided real MAC instructions: To recall, the CMAC $d = a + bc,\ a, b, c, d \in \mathbb{C}$ is defined by

$$\begin{aligned} d_\mathrm{r} &= a_\mathrm{r} + b_\mathrm{r}c_\mathrm{r} - b_\mathrm{i}c_\mathrm{i} \\ d_\mathrm{i} &= a_\mathrm{i} + b_\mathrm{r}c_\mathrm{i} + b_\mathrm{i}c_\mathrm{r}, \end{aligned} \tag{2}$$

with d_r denoting the real and d_i the imaginary part of d. Using the $\mathrm{MAC}(\alpha, \beta, \gamma) = \alpha + \beta\gamma$ operation, (2) resolves to 4 applications of it:

$$\begin{aligned} d_\mathrm{r} &= \mathrm{MAC}(\mathrm{MAC}(a_\mathrm{r}, b_\mathrm{r}, c_\mathrm{r}), -b_\mathrm{i}, c_\mathrm{i}) \\ d_\mathrm{i} &= \mathrm{MAC}(\mathrm{MAC}(a_\mathrm{i}, b_\mathrm{r}, c_\mathrm{i}), b_\mathrm{i}, c_\mathrm{r}). \end{aligned} \tag{3}$$

An alternative to negating b_i is fused multiply-subtraction (MSUB). While RVV includes MSUB for the single-width case (product has same width as factors), it does not for the

This document is a preprint of: V. Razilov, E. Matus and G. Fettweis, "Communications Signal Processing Using RISC-V Vector Extension," in Proceedings of International Wireless Communications and Mobile Computing Conference (IWCMC 2022), Dubrovnik, Croatia, (pp. 690-695), May 2022. DOI:10.1109/IWCMC55113.2022.9824961

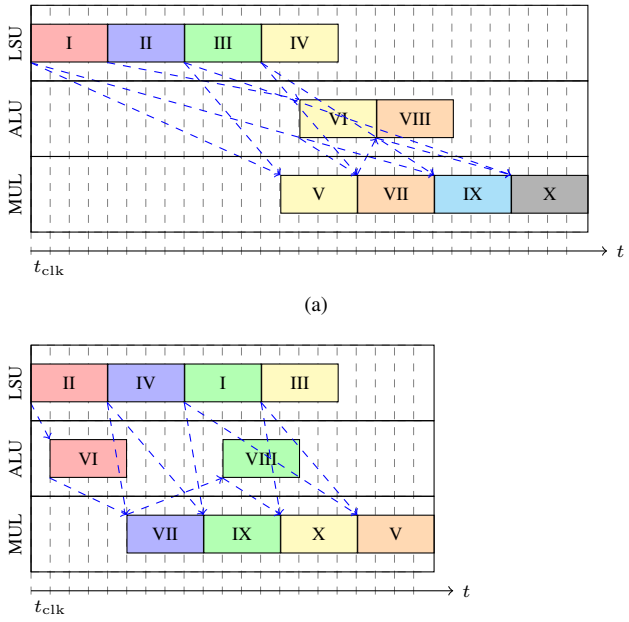

Fig. 4. Simplified swimlane diagram of the chaining of different vector instruction orderings in an in-order vector processor with a load-store unit (LSU), an arithmetic and logic unit (ALU) and an multiplier (MUL). The roman numerals refer to the numbering in Listing 1. Each vector instruction runs for 4 cycles. Instructions in a convoy are colored the same. (a) Naïve approach. (b) Optimized instruction ordering with availability of functional units in mind.

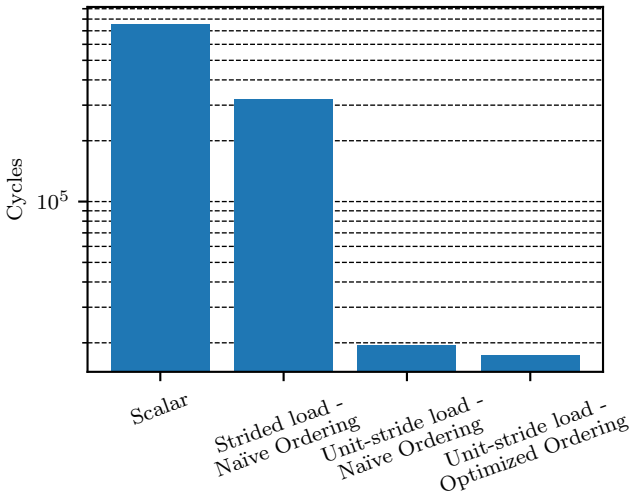

Fig. 5. Cycle count of GFDM ($M = 7$, $K = 512$) on Ara with vector length $\mathrm{VLEN} = n_{\mathrm{v}} w_{\mathrm{d}} = 2048$ and $n_{\mathrm{l}} = 4$ lanes.

widening case (product has twice the width), which is needed here for precision (c.f. Section III-B).

Listing 1 depicts the resulting code for the innermost loop in Fig. 2. The real and imaginary parts are kept in separate memory regions and the data is loaded from memory with vector unit-stride load. In accordance with (3), we temporarily negate the imaginary part of one factor. The cycle overhead of the negations is around 2 %.

Custom complex-number instructions would decrease the instruction count but would also need to be supported by dedicated hardware multipliers: Either by optimized complex-number multipliers (CMUL) besides the real-number multipliers (RMUL) or by additional control logic to rewire 4 RMUL to a single CMUL.

F. Instruction Ordering

High-performance vector processors perform chaining. An instruction that consumes the output of its predecessor is started once the first elements are available—as long as there is no structural hazard between the instructions. In this case, they are said to be in a convoy [25]. Since many vector processors such as Ara execute instructions in order, one needs to keep structural hazards in mind when ordering instructions.

Fig. 4a illustrates this pitfall by showing the chaining that results from the instruction ordering in Listing 1. Instructions I–IV occupy the load-store unit (LSU) and are therefore not in a convoy. Subsequently, instruction V must wait until IV began execution even though it could start already once the first data from III is produced. As a result of this ordering, both, the LSU and the multiplier (MUL), are underutilized.

Fig. 4b presents a better approach where instructions are reordered such that the requested functional unit alternates. Utilization rises and the cycle count for the innermost loop is reduced by almost a third in this model. The performance gain for the entire kernel will be less because of Amdahl's law.

G. Hardware Optimization

When analyzing the execution, we noticed a stall between iterations of the loop in Listing 1 that was caused by an erroneous hazard detection in Ara. A minor patch lowered cycle count by 25 %.

IV. Evaluation

This section contains a cycle count evaluation of the different GFDM implementation variants detailed in section III. For measurement, we use register-transfer level simulations of Ara[1] with a slight modification (c.f. section III). We perform simulations under different configurations of VLEN and n_{l} to study their impact on the execution time.

A. Throughput

The optimizations described in Section III lead to the cycle counts plotted in Fig. 5. The base RISC-V execution, that runs only on Ara's scalar core Ariane, serves as a reference. It uses interleaved arrangement because it is more performant in the scalar case. Vectorizing the code without a change of the memory layout reduces the cycle count by 57 %. A more drastic speedup factor of 39 is achieved if one also changes the arrangement to better suit vector processing. The optimized instruction ordering detailed in Fig. 4b yields another 12 % overall performance gain at no cost.

Figure 6 shows the cycle count for Ara configurations with different number of lanes n_{l} and VLEN. n_{l} can be understood

[1] Version v2.1.0, See https://github.com/pulp-platform/ara/tree/v2.1.0

TABLE I
COMPARISON BETWEEN DIFFERENT GFDM SOLUTIONS.

Platform	M	K	f_{clk} [MHz]	c	f_{s} $\left[\frac{\text{MSymbols}}{\text{s}}\right]$	W [MAC]	Q [B]	π $\left[\frac{\text{MAC}}{\text{cycle}}\right]$	β $\left[\frac{\text{MAC}}{\text{cycle}}\right]$	Π $\left[\frac{\text{MAC}}{\text{cycle}}\right]$	P $\left[\frac{\text{MAC}}{\text{cycle}}\right]$	η [%]
This work—Ariane	7	512	1250	760k	5.9	100k	229k	0.5	8	0.13	0.5	26
This work—Ara $n_{\text{l}} = 4$, VLEN $= 2048$	7	512	1250	17.3k	258	100k	215k	8	16	5.8	7.5	77
This work—Ara $n_{\text{l}} = 16$, VLEN $= 2048$	7	512	1040	4.76	784	100k	215k	32	64	21	30	71
Low-power ASIP [10]	7	512	100	8.96k[a]	40	100k	229k	16	32	11	14	80
FPGA [10], [11]	7	512	150	1.57k[a]	343	—	—	—	—	—	—	—
Vector DSP [8]	7	128	500	504	889	25.1k	53.8k	64	128	56	60	83

[a]Calculated with (5).

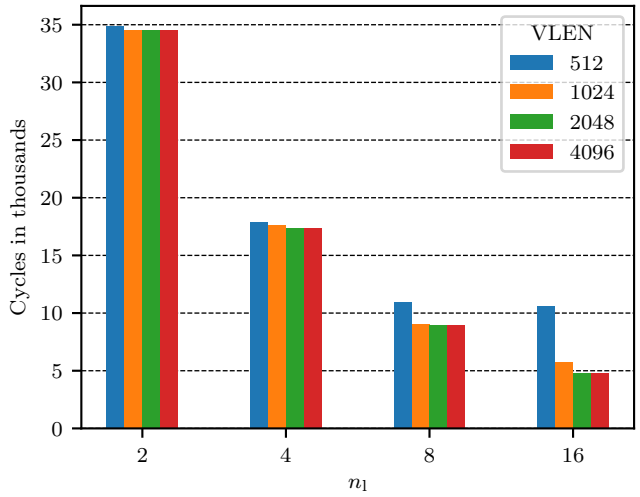

Fig. 6. Cycle count of GFDM ($M = 7$, $K = 512$) on Ara under different vector lengths VLEN $= n_{\text{v}} w_{\text{d}}$ and number of lanes n_{l}.

as the number of parallel computational units, since every lane in Ara contains a MUL and an arithmetic and logic unit (ALU). Both are capable of producing 64 output bits per cycle, meaning that the MUL executes two widening MACs with our chosen data widths. Higher n_{l} result in almost proportionally faster execution of vector instructions, because the majority of the processing time is spent in the RVV processor.

With less loop iterations comes a smaller looping overhead. The number of loop iterations is given by

$$n_{\text{loop}} = \left\lceil \frac{K w_{\text{d}}}{\text{LMUL} \times \text{VLEN}} \right\rceil . \tag{4}$$

Consequently, higher VLEN increase performance, albeit not as much as higher n_{l}. From (4), one can infer why VLEN $= 4096$ brings no further gains: K elements of width w_{d} already fit into 4×2048 bits and the additional VRF space is of no use.

The cycle count c translates into communications symbol rate by

$$f_{\text{s}} = MK \frac{f_{\text{clk}}}{c} \tag{5}$$

where f_{clk} denotes the processor's clock rate—1.25 GHz in Ara's case with $n_{\text{l}} = 4$ [21]. As such, Ara with 4 lanes and 2048 bit in a vector register achieves a symbol rate of $f_{\text{s}} = 258 \ \frac{\text{MSymbol}}{\text{s}}$ and 16 lanes increase the symbol rate to $784 \ \frac{\text{MSymbol}}{\text{s}}$.

B. Utilization

The roofline model [26], [27] is a helpful tool for analyzing processor utilization. It states that the throughput of an algorithm may either be limited by the processor's memory bandwidth β (measured in $\frac{\text{B}}{\text{cycle}}$) or compute bandwidth π ($\frac{\text{OP}}{\text{cycle}}$). The operational intensity I of an algorithm determines which bound is in force and is given by the ratio [27]

$$I = \frac{W}{Q} \tag{6}$$

of the number of operations W and the bytes to be read or written Q. The attainable peak performance is then

$$\Pi = \min(\pi, I\beta). \tag{7}$$

The comparison of the measured performance P to Π reveals an architecture's utilization of its available ressources

$$\eta = \frac{P}{\Pi}. \tag{8}$$

For a fair comparison, we take only real MAC operations into account for the workload:

$$W = 4M^2K. \tag{9}$$

The memory traffic depends on the quantization that was chosen for the input data w_{d} and the result w_{x}:

$$Q = 4M^2Kw_{\text{d}} + 2MKw_{\text{x}}. \tag{10}$$

In this work, w_{x} is 32 bit in the scalar and 16 bit in the vectorized case.

The utilization values of different Ariane/Ara configurations is documented in Table I. As can be seen, the vectorized versions have not only a higher throughput, but also an improved utilization. A higher number of lanes decreases utilization because the scalar instructions' share of the overall runtime rises when vector instructions are executed faster.

C. Comparison

Table I provides a comparison of RVV with other solutions in literature. Ara with 16 lanes outperforms allmost all the other in terms of throughput demonstrating its capability of high-performance number crunching. Only the vector DSP from [8] surpasses Ara. However, when it comes to utilization, Ara is slightly lagging, but yet quite impressive for a general-purpose solution.

V. Conclusion and Outlook

This paper demonstrated a software implementation of GFDM on a RVV processor. It traced the porting procedure from algorithm analysis over data structure selection to programming. We draw attentions to pitfalls involved and exemplified some beneficial features of RVV. The characteristics of RVV allow for seamless programming of a wide range of procesors with different configurations. The vectorized version achieves a peak throughput of 784 $\frac{\text{MSymbol}}{\text{s}}$ and a 60 times speedup compared to the scalar version. However, it is lagging in performance and utilization when compared to more specialized DSPs.

In future work, we will investigate how the performance of communications signal processing tasks on RVV can be brought closer to DSPs—both, by changes to the vector processor's microarchitecture and by ISA enhancements. Possible measures may be implementation of segment load, improved scheduling, and complex-arithmetic ISA extensions.

Acknowledgment

This work was funded in part by the German Federal Ministry of Education and Research (BMBF) in the project "E4C" (project number 16ME0426K).

References

[1] M. Jian and R. Liu, "Baseband signal processing for terahertz: waveform design, modulation and coding," in *IEEE Int. Wireless Communications and Mobile Computing Conf. (IWCMC)*, Harbin City, China, Jun. 2021, pp. 1710–1715.

[2] G. P. Fettweis and H. Boche, "6G: The personal tactile internet - and open questions for information theory," *IEEE BITS Inf. Theory Mag.*, early access, Oct. 2021, doi: 10.1109/MBITS.2021.3118662.

[3] S. Shahabuddin, A. Mämmelä, M. Juntti, and O. Silvén, "ASIP for 5G and beyond: opportunities and vision," *IEEE Trans. Circuits Syst., II, Exp. Briefs*, vol. 68, no. 3, pp. 851–857, Jan. 2021.

[4] A. Waterman and K. Asanović, Eds., *The RISC-V Instruction Set Manual, Volume I: User-Level ISA Version 2.1*, Document Version 20191213, RISC-V Foundation, Dec. 2019.

[5] *RISC-V "V" Vector Extension*, Version 1.0, Sep. 2021. [Online]. Available: https://github.com/riscv/riscv-v-spec/releases/download/v1.0/riscv-v-spec-1.0.pdf

[6] D. Dabbelt, C. Schmidt, E. Love, H. Mao, S. Karandikar, and K. Asanovic, "Vector processors for energy-efficient embedded systems," in *3rd ACM Int. Workshop Many-core Embedded Systems*, Seoul, Republic of Korea, Jun. 2016, pp. 10–16.

[7] N. Michailow *et al.*, "Generalized frequency division multiplexing for 5th generation cellular networks," *IEEE Trans. Commun.*, vol. 62, no. 9, pp. 3045–3061, Sep. 2014.

[8] S. A. Damjancevic, E. Matus, D. Utyansky, P. van der Wolf, and G. Fettweis, "Towards GFDM for handsets - efficient and scalable implementation on a vector DSP," in *IEEE 19th Vehicular Technology Conf. (VTC2019-Fall)*, Honolulu, HI, USA, Sep. 2019, pp. 1–7.

[9] S. A. Damjancevic, E. Matus, D. Utyansky, P. van der Wolf, and G. Fettweis, "From challenges to hardware requirements for wireless communications reaching 6G," in *Multi-processor System-on-Chip 2 - Applications*, L. Andrade and F. Rousseau, Eds., Hoboken, NJ, USA: Wiley, 2020, pp. 3–29.

[10] R. Wittig, S. A. Damjancevic, E. Matus, and G. P. Fettweis, "General multicarrier modulation hardware accelerator for the internet of things," in *IEEE Global Communications Conf. (GLOBECOM)*, Waikaloa, HI, USA, Dec. 2019, pp. 1–6.

[11] M. Danneberg *et al.*, "Universal waveforms processor," in *European Conf. Networks and Communications (EuCNC)*, Ljubljana, Slovenia, Jun. 2018, pp. 357–362.

[12] A. Darghouti, A. Khlifi, and B. Chibani, "Performance analysis of 5G waveforms over fading environments," in *IEEE Int. Wireless Communications and Mobile Computing Conf. (IWCMC)*, Harbin City, China, Jun. 2021, pp. 2182–2187.

[13] M. Matthé, L. Mendes, I. Gaspar, D. Zhang, and G. Fettweis, "Precoded GFDM transceiver with low complexity time domain processing," *EURASIP J. Wireless Commun. and Netw.*, vol. 2016, May 2016, Art. no. 138.

[14] A. Farhang, N. Marchetti, and L. E. Doyle, "Low-complexity modem design for GFDM," *IEEE Trans. Signal Process.*, vol. 64, no. 6, Mar. 2016, pp. 1507–1518.

[15] H. B. Amor, C. Bernier, Z. Přikryl, "A RISC-V ISA extension for ultra-low power IoT wireless signal processing,", *IEEE Trans. Comput.*, early access, Mar. 2021, doi: 10.1109/TC.2021.3063027.

[16] M. Tourres, C. Chavet, B. Le Gal, J. Crenne, and P. Coussy, "Extended RISC-V hardware architecture for future digital communication systems," in *IEEE 4th 5G World Forum (5GWF)*, Montreal, QC, Canada, Nov. 2021, pp. 224–229.

[17] H. Saidi, M. Turki, Z. Marrakchi, A. Obeid, and M. Abid, "Implementation of Reed Solomon encoder on low-latency embedded FPGA in flexible SoC based on ARM processor," in *IEEE Int. Wireless Communications and Mobile Computing Conf. (IWCMC)*, Limassol, Cyprus, Jun. 2020, pp. 1347–1352.

[18] *RISC-V "P" Extension Proposal*, Version 0.9.11-draft-20211209, Dec. 2021. [Online]. Available: https://github.com/riscv/riscv-p-spec/blob/master/P-ext-proposal.pdf

[19] N. Stephens *et al.*, "The ARM scalable vector extension," *IEEE Micro*, vol. 37, no. 2, 2017, pp. 26–39, Mar./Apr. 2017.

[20] A. Pohl, M. Greese, B. Cosenza, and B. Juurlink, "A performance analysis of vector length agnostic code," in *Int. Conf. High Performance Computing & Simulation (HPCS)*, Dublin, Ireland, Jul. 2019, pp. 159–164.

[21] M. Cavalcante, F. Schuiki, F. Zaruba, M. Schaffner, and L. Benini, "Ara: a 1-GHz+ scalable and energy-efficient RISC-V vector processor With multiprecision floating-point support in 22-nm FD-SOI," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.* , vol. 28, no. 2, pp. 530–543, Feb. 2020.

[22] K. Patsidis, C. Nicopoulos, G. C. Sirakoulis, and G. Dimitrakopoulos, "RISC-V^2: a scalable RISC-V vector processor," in *IEEE Int. Symp. Circuits and Systems (ISCAS)*, Seville, Spain, Oct. 2020, pp. 1–5.

[23] M. Johns and T. J. Kazmierski, "A minimal RISC-V vector processor for embedded systems," in *2020 Forum Specification and Design Languages (FDL)*, Kiel, Germany, Sep. 2020, pp. 1–4.

[24] M. Platzer and P. Puschner, "Vicuna: a timing-predictable RISC-V vector coprocessor for scalable parallel computation," in *33rd Euromicro Conference Real-Time Systems (ECRTS)*, Dagstuhl, Germany, Jun. 2021, pp. 1:1–1:18.

[25] J. L. Hennesy and D. A. Patterson, *Computer Architecture - A Quantitative Approach*, 6th Ed., San Mateo, CA, USA: Morgan Kaufmann, 2019.

[26] S. Williams, A. Waterman, and D. Patterson, "Roofline: an insightful visual performance model for multicore architectures," *Commun. ACM*, vol. 52, no. 4, Apr. 2009, pp. 65–76.

[27] G. Ofenbeck, R. Steinmann, V. Caparros, D. G. Spampinato, M. Püschel, "Applying the roofline model," in *IEEE Int. Symp. Perf. Anal. Syst. Softw. (ISPASS)*, Monterey, CA, USA, Mar. 2014, pp. 75–85.