Variable Precision Capabilities in RISC-V Processors

Tiago Trevisan Jost[1], Andrea Bocco[1], Yves Durand[1], Christian Fabre[1], Florent De Dinechin[2], Anca Molnos[1], Albert Cohen[3]
[1]CEA, LETI, Univ. Grenoble Alpes, 38000 Grenoble, France; [2]INSA, Lyon, France; [3]Google, Inc.
Email: andrea.bocco@cea.fr, tiago.trevisanjost@cea.fr

***Abstract*—This work proposes to extend RISC-V with Variable Precision (VP) Floating-Point (FP) capabilities to accelerate scientific computing applications. It adopts the UNUM type I FP format in main memory to overcome the limitation of the IEEE 754 standard. Our work comprises: 1/ a VP FP RISC-V coprocessor; 2/ a RISC-V ISA extension for the unit, 3/ and a programming model to support VP floats in C/C++. Results have shown that our system can be more than 100x faster than the MPFR library when executing basic arithmetic operations.**

I. Introduction

Current computing systems for scientific applications extensively use the IEEE 754 standard for Floating-Point (FP) representation [1]. However, scientific computing applications (e.g. ab initio simulations [2] and small scale physical modeling [3]) require large and adaptive precision during the course of their processing (up to hundreds of digits). These applications suffer from cancellation and rounding errors, and increasing computation precision may significantly increase stability, convergence, and compensate the conditioning issues. Therefore, users often rely on multi-precision libraries, such as MPFR [4] or GMP [5], to achieve satisfying results accuracy at the cost of increasing the kernel execution time.

This work proposes Variable Precision (VP) FP hardware and software support in a RISC-V environment to have better control of accuracy for FP applications. It comprises a RISC-V [6] coprocessor that supports a VP format for FP computation, an ISA extension, and a programming model to use the system. This work is an update of the one presented at the 2018 RISC-V Workshop at Chennai, India. The main improvements of this work are: 1/ an update to the VP ISA extension with instructions that can operate with scalars and intervals, 2/ a RISC-V MPFR port to serve as baseline in experiments, 3/ and a comparison between the ISA instruction and the MPFR library. To the best of our knowledge, this work presents the first integrated HW-SW solution for VP computation for scientific applications. Some previous VP hardware implementations [7, 8] have been proposed but no emphasis had been given to software integration.

In this paper: Section II describes the used VP representation, and introduces the coprocessor unit. Section III describes the main properties of the ISA extension. Section IV covers the programming model. Section V presents experimental results and Section VI concludes this work.

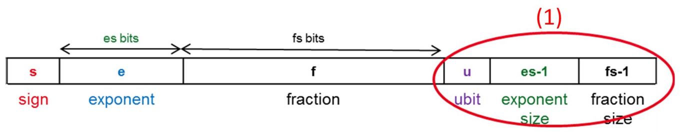

Fig. 1. UNUM Floating-Point format

II. The coprocessor unit and UNUM format

We implemented a VP FP coprocessor unit which supports basic scalar and interval arithmetic operations [9] (+,-,*), We chose the UNUM format [10] (Fig. 1) in memory. It is encoded as sign, exponent, mantissa and self-descriptive fields ((1) on Fig. 1). Those fields encode the length of the *e* and *f* fields and provide support for interval arithmetic [10]. Our hardware unit (Fig. 2) is tightly coupled to a RISC-V Rocket Chip processor (① on Fig. 2), through the RoCC interface ②.

In addition to augment the data size in memory, the UNUM format tends to generate misaligned accesses which degrade access time. Therefore, the VP arithmetic unit requires a custom "Load and Store" unit ③ which realigns data in memory ⑤ and handles misaligned memory accesses.

Our coprocessor hosts a VP register file (vRF) in the internal scratchpad ④. It contains 32 registers (v0-v31) which hold VP intervals. Each interval endpoint mantissa is organized in eight 64-bit chunks (for a maximum of 512 bits). A descriptor is used to encode how many mantissa chunks are actually used for each interval endpoint. Its value depends on the number of bits of precision specified by the user at code level. By doing so, we maintain operations in the coprocessor fast and coherent to the precision set by the user. The compiler is in charge of mapping VP variables into the scratchpad

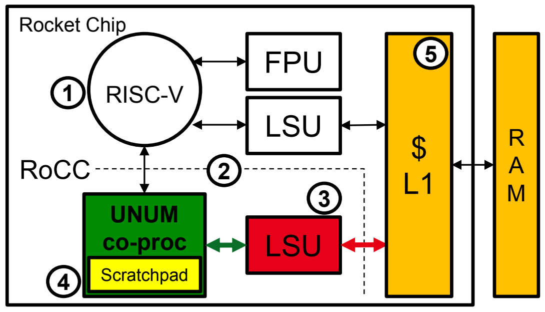

Fig. 2. Architecture of the host/coprocessor couple

TABLE I
VARIABLE PRECISION ISA EXTENSION

	31 25	24 20	19 15	14	13	12	11 7	6 0
⓪	func7	rs2	rs1	xd	xs1	xs2	rd	opcode
	7	5	5	1	1	1	5	7
①	SUSR	unused	Xs1	0	1	0	unused	OP-CUST
②	LUSR	unused	unused	1	0	0	Xd	OP-CUST
③	MOV_G2G	unused	gRs1	0	0	0	gRd	OP-CUST
④	MOVLL/MOVLR	unused	gRs1	0	0	0	gRd	OP-CUST
⑤	MOVRL/MOVRR	unused	gRs1	0	0	0	gRd	OP-CUST
⑥	MOV_X2G	#imm5	Xs1	0	1	0	gRd	OP-CUST
⑦	MOV_G2X	#imm5	gRs2	1	0	0	Xd	OP-CUST
⑧	GCMP	gRs2	gRs1	1	0	0	Xd	OP-CUST
⑨	GADD/GSUB/GMUL	gRs2	gRs1	0	0	0	gRd	OP-CUST
⑩	GGUESS/GRADIUS	unused	gRs1	0	0	0	gRd	OP-CUST
⑪	LDU/LDUB	unused	Xs1	0	1	0	gRd	OP-CUST
⑫	STUL/STUB	gRs2	Xs1	0	1	0	unused	OP-CUST
⑬	LDU_NEXT/LDUB_NEXT	gRs2	Xs1	1	1	0	Xd	OP-CUST
⑭	STUL_NEXT/STUB_NEXT	gRs2	Xs1	1	1	0	Xd	OP-CUST

Example 1 Usage of the variable precision unit

```
vpfloat(16,256) factorial(vpfloat(16,256) k) {
    vpfloat(16,256) fact = 1;
    for (int i = 1; i < k+1; ++i) {
        fact *= i;
    }
    return fact;
}
```

memory. Supporting VP computation in hardware complicates the coprocessor pipeline design since hardware operations on multiple chunks mantissas have to be treated iteratively.

III. ISA EXTENSION FOR VP

The RISC-V ISA is extended to map the coprocessor instructions on a dedicated set of opcodes. Table I shows the coprocessor instructions to manipulate VP numbers. They make use of the coprocessor vRF described in section II.

Instructions are divided into four groups: internal status register manipulation (①-②); register move operations (③-⑦) that cover moves between coprocessor and main processor RFs, and between VP registers; arithmetic operation (⑧-⑩) and memory-related operations (⑪-⑭).

IV. PROGRAMMING MODEL FOR THE VP UNIT

Exploring new functionalities to the hardware also requires a substantial effort at the software side. Our proposal includes a programming model to support the use of Variable Precision (VP) Floating-Point (FP) in C/C++.

Along with the hardware proposal for VP computing, we propose the new `vpfloat` C data type. With it, the user can declare VP FP variables by specifying the maximum lengths of the exponent and mantissa fields.

Example 1 shows how to calculate the factorial of numbers using `vpfloat` in C. This algorithm tends to generate numbers with high orders of magnitude, so it is fitted as a practical example of how VP can be used in code.

We have extended LLVM [11] with an initial support for the `vpfloat` type. Compiler is able to identify `vpfloat` variables and to assign exponent and precision to the middle-end representation. It also supports the coprocessor ISA, generating VP FP instructions. Once that the code is compiled, the RISC-V GNU Assembler and Linker were expanded to generate executable code for the coprocessor extension.

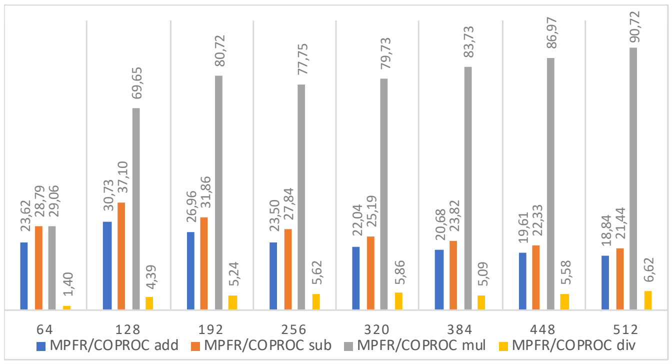

Fig. 3. MPFR vs. coprocessor: single operation clock cycle comparison

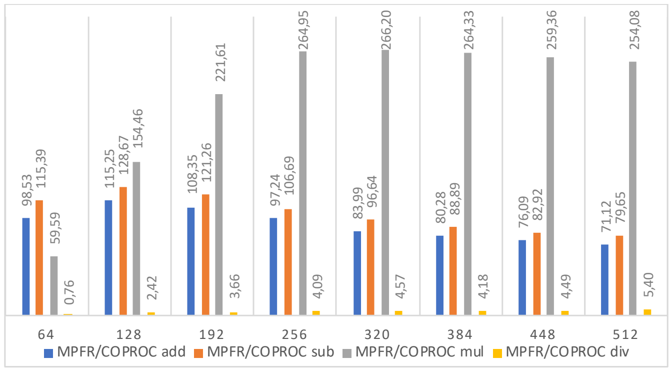

Fig. 4. MPFR vs. coprocessor: series of 10 operation clock cycle comparison

V. EXPERIMENTAL RESULTS

Fig. 3 and Fig. 4 show the speedup achieved by our coprocessor in comparison to the MPFR library on an FPGA. We have tested the basic arithmetic operators (+, -, *, /): 1/ with a single operation (Fig. 3) 2/ and with ten consecutive independent operations (Fig. 4). Results show that operations between 64 and 512 bits of precision can achieve speedups of more than 100x for addition, subtraction and multiplication. We implement the division by software, hence it can not reach the same speedup ratio.

VI. CONCLUSION

This work aimed at increasing accuracy on scientific applications by proposing a RISC-V VP FP computing system. The work included: a coprocessor, an ISA, a programming model, and compiler support. Measurements showed that speedups of more than 100x are possible to be achieved compared to the MPFR software library.

As this is a work-in-progress, we envision future work including: 1/ Implement the system in silicon; 2/ Improve compiler support; 3/ Validate and experiment with real-life applications, such as the three-body problem [12] and computational fluid dynamics [13].

REFERENCES

[1] *IEEE Standard for Floating-Point Arithmetic*. IEEE 754-2008, also ISO/IEC/IEEE 60559:2011. Aug. 2008. DOI: 10.1109/IEEESTD.2008.4610935.

[2] David H Bailey and Jonathan M Borwein. “High-precision arithmetic in mathematical physics”. In: *Mathematics* (2015), pp. 337–367.

[3] David H Bailey. “High-precision floating-point arithmetic in scientific computation”. In: *Computing in science & engineering* 7.3 (2005), pp. 54–61.

[4] Laurent Fousse et al. “MPFR: A Multiple-precision Binary Floating-point Library with Correct Rounding”. In: *ACM Trans. Math. Softw.* 33.2 (June 2007). ISSN: 0098-3500. DOI: 10.1145/1236463.1236468. URL: http://doi.acm.org/10.1145/1236463.1236468.

[5] Torbjörn Granlund and the GMP development team. *GNU MP: The GNU Multiple Precision Arithmetic Library*. Version 5.0.5. 2012. URL: https://gmplib.org/.

[6] “RISC-V Foundation - Instruction Set Architecture (ISA)”. In: URL: https://riscv.org.

[7] F. Glaser et al. “An 826 MOPS, 210uW/MHz Unum ALU in 65 nm”. In: *2018 IEEE International Symposium on Circuits and Systems (ISCAS)*. May 2018, pp. 1–5.

[8] J. Hou et al. “Enhancing Precision and Bandwidth in Cloud Computing: Implementation of a Novel Floating-Point Format on FPGA”. In: *IEEE CSCloud*. June 2017, pp. 310–315.

[9] A. Bocco, Y. Durand, and F. De Dinechin. “Hardware support for UNUM floating point arithmetic”. In: *PRIME*. IEEE. 2017, pp. 93–96.

[10] John L Gustafson. *The End of Error: Unum Computing*. Chapman and Hall/CRC, 2015.

[11] C. Lattner and V. Adve. “LLVM: a compilation framework for lifelong program analysis transformation”. In: *CGO*. 2004, pp. 75–86.

[12] Christian Marchal. *The three-body problem*. Elsevier, 2012.

[13] S. Che et al. “Rodinia: A benchmark suite for heterogeneous computing”. In: *2009 IEEE International Symposium on Workload Characterization (IISWC)*. Oct. 2009, pp. 44–54. DOI: 10.1109/IISWC.2009.5306797.