VP Float: First Class Treatment for Variable Precision Floating Point Arithmetic

Tiago Trevisan Jost, Yves Durand, Christian Fabre, Albert Cohen, Frédéric Pétrot

VP Float: First Class Treatment for Variable Precision Floating Point Arithmetic

Tiago Jost
Yves Durand
Christian Fabre
tiago.trevisanjost@cea.fr
Univ. Grenoble Alpes
CEA, LIST, Grenoble, France

Albert Cohen
albertcohen@google.com
Google, Paris, France

Frédéric Pétrot
frederic.petrot@univ-grenoble-alpes.fr
Univ. Grenoble Alpes, CNRS,
Grenoble INP†, TIMA, Grenoble, France

ABSTRACT

Optimizing compilers for high performance computing only support IEEE 754 floating-point (FP) types and applications needing higher precision involve cumbersome memory management and calls to external libraries. We introduce an extension of the C type system to represent variable-precision FP arithmetic, supporting both static and dynamically variable precision. We design and implement a compilation flow bridging the abstraction gap between this type system and hardware FP instructions or software libraries. We demonstrate the effectiveness of our solution by enabling the full range of LLVM optimizations and leveraging two backend code generators: one for the ISA of a variable precision FP arithmetic coprocessor, and one for the MPFR multi-precision FP library. Both targets support the static and dynamically adaptable precision of our type system. On the PolyBench suite, our optimizing compilation flow targeting MPFR is shown to outperform the Boost programming interface for the MPFR library.

CCS CONCEPTS

• **Software and its engineering → Source code generation**; *Imperative languages*; *Data types and structures*; *Dynamic compilers.*

KEYWORDS

Floating point arithmetic, compiler optimization, LLVM, MPFR

ACM Reference Format:
Tiago Jost, Yves Durand, Christian Fabre, Albert Cohen, and Frédéric Pétrot. 2020. VP Float: First Class Treatment for Variable Precision Floating Point Arithmetic. In *Proceedings of the 2020 International Conference on Parallel Architectures and Compilation Techniques (PACT '20), October 3–7, 2020, Virtual Event, GA, USA.* ACM, New York, NY, USA, 2 pages. https://doi.org/10.1145/3410463.3414660

1 INTRODUCTION

Floating-point (FP) computation has been around long before its standardization, but the adoption of the IEEE 754 standard in 1985 [10] along with the progress of integration technology made hardware FP units ubiquitous. While hardware support is paramount to the performance of numerical applications, compilers play a major role in leveraging these units efficiently. Compiler optimizations handle FP representations supported by common hardware, at best the 16, 32, 64, 80 and 128 bits IEEE formats as well as bf16. However, a growing number of applications [2, 6] are better suited to operate on different formats.

```
1  void axpy100(int N, vpfloat<mpfr, 16, 100> alpha,
2               vpfloat<mpfr, 16, 100> *X,
3               vpfloat<mpfr, 16, 100> *Y) {
4      for (unsigned i = 0; i < N; ++i)
5          Y[i] = alpha * X[i] + Y[i];
6  }
7
8  void vaxpy(unsigned prec, int N,
9             vpfloat<mpfr, 16, prec> alpha,
10            vpfloat<mpfr, 16, prec> *X,
11            vpfloat<mpfr, 16, prec> *Y) {
12     for (unsigned i = 0; i < N; ++i)
13         Y[i] = alpha * X[i] + Y[i];
14 }
```

Listing 1: *axpy* kernel with *mpfr* type

Finer control of exponent and precision sizes allows the numerical analyst to explore suitable trade-offs between accuracy and execution time (and/or energy consumption). This goal can be achieved through the use of external libraries, yet multi-precision code is difficult to write and maintain. More than the performance gap, the productivity gap of variable precision FP arithmetic makes it unaccessible to its main potential users.

We address this challenge through a variable precision FP type system and language extension of standard C through the introduction of a template type named `vpfloat`. The syntax for this type system captures most of the expressiveness needed by numerical analysts while enabling highly efficient in-place execution, stack allocation and the full range of compiler optimizations expected for the C language. This is made possible by extending the LLVM intermediate representation [7], allowing classical compiler optimizations to operate on multi-precision FP types with few modifications. In particular, we enable multiple FP formats to coexist in a single numerical kernel with full procedural abstraction. For each format that supports it, we also enable computations over multiple variables of different precision and memory footprints, including dynamically-varying precision and footprints. This allows to explore multiple numerical configurations within a single program source or even across successive iterations of a tuning or convergence loop.

† Institute of Engineering Univ. Grenoble Alpes.

PACT '20, October 3–7, 2020, Virtual Event, GA, USA

ACM ISBN 978-1-4503-8075-1/20/10.
https://doi.org/10.1145/3410463.3414660

2 GENERIC VP TYPES

Every VP variable declaration must provide a *type* attribute which defines if subsequent attributes are needed and which information they carry. Attributes are specified in the following order: *type*, *exponent*, *precision*, and *size*. With the exception of *type*, `vpfloat` attributes can all be defined with integral constant literals or identifiers. This generic type supports not only constant-size but also dynamically-sized types, which is always a technical hurdle in languages with unmanaged memory like C, and its associated intermediate representations and ABIs.

We designed and implemented a full compilation flow in LLVM supporting (1) `mpfr` types, which hold the number of bits of exponent and mantissa in the second and third fields, respectively; and (2) `unum` [5] types with second and third fields, *ess* and *fss*, respectively, with an optional *size* field that holds the maximum number of bytes used to represent the number. Listing 1 shows the implementation of the *axpy* kernel, a level-1 BLAS [8] routine operating on vectors, for constant and dynamically-sized *mpfr* types.

3 EXPERIMENTAL RESULTS

We demonstrate the effectiveness of our type system and compiler implementation by comparing the variable-precision MPFR type `vpfloat<mpfr, ...>` with the Boost library for multi-precision. Both approaches rely on the MPFR library and execute code with identical precision [3]. We compiled the PolyBench suite version 4.1 [9] at optimization level -O3, and enabling or disabling Polly's polyhedral loop nest optimizations [4]. The *nussinov* benchmark results are missing due to erroneous computations (NaN) in the Boost baseline, we are investigating the issue. Overall, results show an average speedup of 1.70× for the Intel Xeon E5-2637 with 128GB of RAM.

To evaluate the portability of our approach, we also demonstrate our LLVM implementation targeting a coprocessor for a RISC-V Rocket core accelerating FP arithmetic in the UNUM format [1]. Unfortunately we hit hardware bugs when executing some benchmarks: *gesummv* and *adi* failed to run when compiled with Polly and 3 more benchmarks failed at the highest precision with Polly (*3mm*, *ludcmp*, *nussinov*). This is due to an issue in the co-processor memory subsystem that we were not able to address at this time. Nevertheless, we were able to achieve at the highest precision (150 digits), speedups of 18.03× and 27.58× for -O3, and -O3 + Polly, respectively, when using our `vpfloat<mpfr, ...>` as baseline.

4 CONCLUSION

We propose an extension to the C type system to operate on variable precision FP formats. Our extension supports FP arithmetic of arbitrary representation whose precision and exponent size can be configured at compilation time or runtime. We demonstrate the productivity benefits of our programming model and its ability to leverage the full range of optimizations of LLVM. Experiments on the PolyBench suite yield strong speedups at all optimization levels in comparison to the Boost Multi-precision library for MPFR.

REFERENCES

[1] Andrea Bocco, Yves Durand, and Florent De Dinechin. 2019. SMURF: Scalar Multiple-precision Unum Risc-V Floating-point Accelerator for Scientific Computing. In *Proceedings of the Conference for Next Generation Arithmetic*. 1–8.

[2] Erin Carson and Nicholas Higham. 2018. Accelerating the Solution of Linear Systems by Iterative Refinement in Three Precisions. *SIAM Journal on Scientific Computing* 40 (Jan. 2018), A817–A847. https://doi.org/10.1137/17M1140819

[3] Laurent Fousse, Guillaume Hanrot, Vincent Lefèvre, Patrick Pélissier, and Paul Zimmermann. 2007. MPFR: A Multiple-precision Binary Floating-point Library with Correct Rounding. *ACM Trans. Math. Softw.* 33, 2, Article 13 (June 2007).

[4] Tobias Grosser, Hongbin Zheng, Raghesh Aloor, Andreas Simbürger, Armin Größlinger, and Louis-Noël Pouchet. 2011. Polly – Polyhedral optimization in LLVM. In *Proceedings of the First International Workshop on Polyhedral Compilation Techniques (IMPACT)*. 1–6.

[5] John L. Gustafson. 2017. *The end of error: UNUM computing*. 416 pages.

[6] Nicholas J. Higham. 1996. *Accuracy and Stability of Numerical Algorithms*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA.

[7] Chris Lattner and Vikram Adve. 2004. LLVM: A Compilation Framework for Lifelong Program Analysis & Transformation. In *Proceedings of the International Symposium on Code Generation and Optimization*. 75–86.

[8] Chuck L Lawson, Richard J. Hanson, David R Kincaid, and Fred T. Krogh. 1979. Basic linear algebra subprograms for Fortran usage. *ACM Transactions on Mathematical Software (TOMS)* 5, 3 (1979), 308–323.

[9] Louis-Noël Pouchet et al. 2012. PolyBench: The polyhedral benchmark suite. http://www.cs.ucla.edu/pouchet/software/polybench.

[10] Dan Zuras et al. 2008. *IEEE standard for floating-point arithmetic*. 70 pages.

This work was partially funded by the French *Agence nationale de la recherche (ANR)* for project IMPRENUM under grant n° ANR-18-CE46-0011.

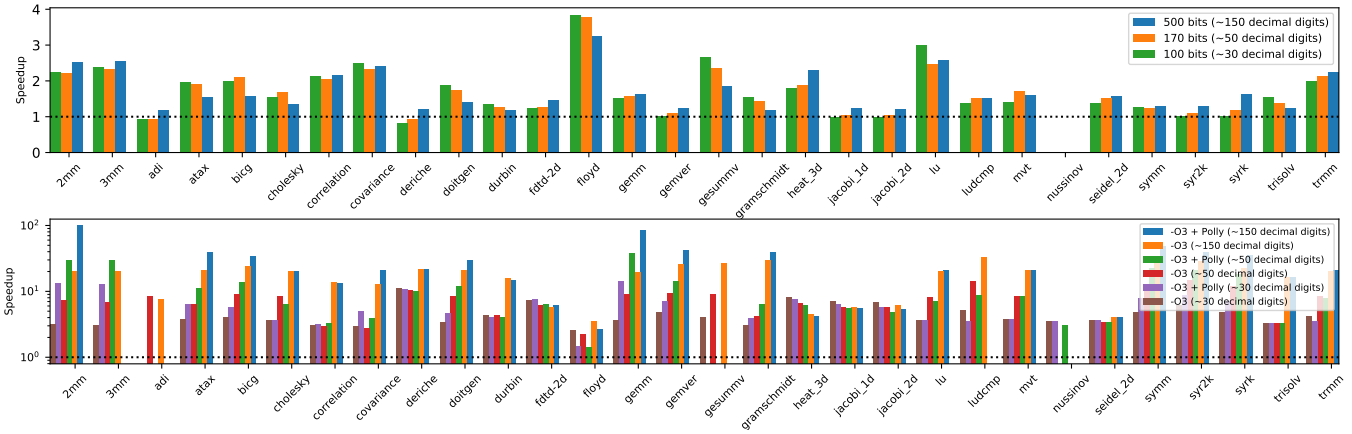

Figure 1: Speedup of (1) `vpfloat<mpfr, ...>` over the Boost library for multi-precision on x86, and (2) of `vpfloat<unum, ...>` over `vpfloat<mpfr, ...>` on an UNUM-accelerated RISC-V processor — PolyBench suite