Automatic exploration of reduced floating-point representations in iterative methods

Yohan Chatelain[1,4], Eric Petit[5,4], Pablo de Oliveira Castro[1,4], Ghislain Lartigue[3], and David Defour[2,4]

[1]Université de Versailles Saint-Quentin-en-Yvelines, Li-PaRAD
[2]Université de Perpignan Via Domitia, LAMPS
[3]Normandie Université, CORIA – CNRS
[4]Exascale Computing Research, ECR
[5]Intel Corporation

Abstract. With the ever-increasing need for computation of scientific applications, new application domains, and major energy constraints, the landscape of floating-point computation is changing. New floating-point representation formats are emerging and there is a need for tools to simulate their impact in legacy codes. In this paper, we propose an automatic tool to evaluate the effect of adapting the floating point precision for each operation over time, which is particularly useful in iterative schemes. We present a backend to emulate any IEEE-754 floating-point operation in lower precision. We tested the numerical errors resilience of our solutions thanks to Monte Carlo Arithmetic and demonstrated the effectiveness of this methodology on YALES2, a large Combustion-CFD HPC code, by achieving 28% to 67% reduction in communication volume by lowering precision.

1 Introduction

Representing infinite real numbers on a finite machine format exposes complex trade-off: IEEE-754 floating-point (FP) single and double are widely used standardized formats which have empirically proved their efficiency. Today's numerical algorithms conveniently rely on double precision FP format. Numerical formats outside IEEE-754 single and double FP are largely unexplored in hardware and software design for HPC.

With new application domains such as machine learning, FP arithmetic used in general purpose processor is entering a new burst of evolution as the IEEE-754 single and double FP are not necessarily the best choices for these problems. Novel hardware and software solutions are explored: variable precision formats (e.g. on FPGAs [15]), new vector instructions such (e.g. Intel Vector Neural Network Instruction), novel FP representations (e.g. BFloat [11], posits [18], fpanr [12]), and libraries for approximate computing [22].

In this paper we propose a methodology to explore alternative representations of hardware and software FP. Our first application is the widely used Computational Fluid Mechanics (CFD) solver YALES2 operated in many recent HPC simulations [25, 3, 6, 4, 17].

Variable precision can be harnessed at the application, compiler, and architecture levels. Our exploration goes beyond traditional mixed precision (single and double format). Due to hardware memory constraints of general purpose processors, the target format can only be represented with 8,16,32 or 64 bits. However, accuracy can be lowered to any given precision to limit the cost of computation and energy. Finally, to go further in the optimization, one may use architectural support, like FPGAs, where precision is fine-tuned up to the bit level [15].

The contributions of this paper are:

- An empirical methodology integrated in Verificarlo [7, 14] to automatically lower precision in iterative algorithms while maintaining an user defined accuracy criterion.
- A VPREC backend to Verificarlo to emulate variable FP representations with a $[1, 11]$-bits exponent and $[0, 52]$-bits pseudo-mantissa.
- Integration of the temporal dimension in the exploration of FP precision. Precisions are not fixed for a variable or code section, but adapt as time passes.
- A validation step based on stochastic arithmetic [37] to increase robustness to rounding and cancellation errors.
- An evaluation of the validity and the performance gains of the proposed methodology on an actual implementation demonstrating runtime savings in an industrial use case.

2 Motivating example

Let us consider as a simple example the Newton–Raphson method which finds the root $x^\star$ of a real-valued function f such that $f(x^\star) = 0$. Starting from an initial guess x_0, the method iteratively computes $x_{k+1} = x_k \quad f(x_k)/f^0(x_k)$ until the relative error between two successive iterations is below a given threshold. At iteration k, the error ϵ_k and the number of significant digits s_k^β in base β are defined as:

$$\epsilon_k = \left| \frac{x^\star \quad x_{k+1}}{x^\star} \right| \qquad\qquad s_k^\beta = \quad \log_\beta(\epsilon_k)$$

The speed of convergence of this method is quadratic [20], which means that the number of significant digits doubles at each iteration. Table 1 shows the evolution of x_k and s_k when computing the inverse of π ($f(x) = 1/x \quad \pi$) with a stopping threshold set to $10^{\ 15}$. The insignificant digits are highlighted in gray.

In the first iterations, most digits are incorrect, hinting that a low precision for the evaluation of f is enough. We used the methodology proposed in this paper and described in section 3 to explore the impact of lowering the precision with the VPREC backend for Verificarlo presented in section 3.1.

Our methodology automatically finds for each iteration k, a reduced precision p_k; the p_k are chosen such that the overall convergence speed is not degraded.

k	x_{k+1}	s_k^{10}	s_k^2
0	0.0690266447076745	0.11	0.37
1	0.1230846130203958	0.21	0.70
2	0.1985746566605835	0.43	1.43
3	0.2732703639721015	0.84	2.79
4	0.3119369815109966	1.79	5.95
5	0.3181822938100336	3.40	11.3
6	0.3183098350392471	6.79	22.6
7	0.3183098861837825	13.6	45.2
8	0.3183098861837907	15.6	51.8
9	0.3183098861837907	15.6	51.8

```
double newton(double x0) {
  double x_k, x_k1=x0, b=PI;
  do {
    x_k  = x_k1;
    x_k1 = x_k*(2-b*x_k);
  }while (fabs((x_k1-x_k)/x_k)
                 >= 1e-15);
  return x_k1;
}
```

Table 1. (*left*)Convergence speed of Newton-Raphson for the computation of the inverse of using the IEEE-754 `binary64` format (*right*). The stopping threshold is 10^{-15}. Highlighted digits in gray are non significant.

Figure 1 shows on the left the p_k's value found by VPREC. The plot at right compares the convergence speed of the standard `binary64` representation and the VPREC configuration. Both versions converge within the 10^{-15} threshold in nine iterations.

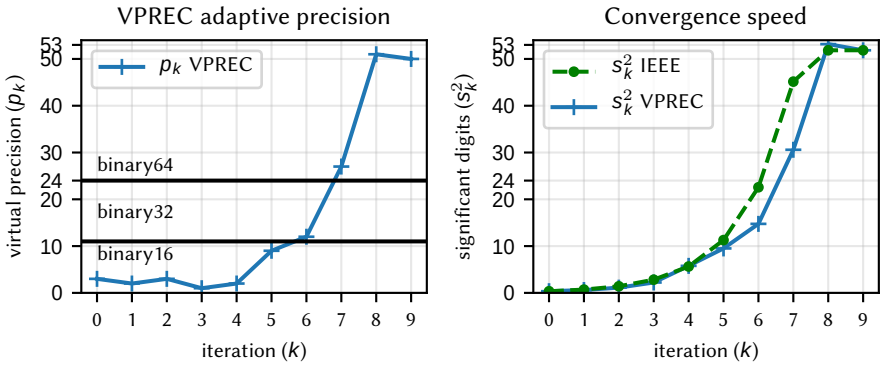

Fig. 1. *(left)* Precision found with VPREC for Newton-Raphson. *(right)* Both the standard IEEE-754 binary64 version and the VPREC low precision configuration converge to 10^{-15} in nine iterations.

Since the first seven iterations require less than 24 bits of precision, they can be executed in single precision (IEEE-754 `binary32`). To validate the solution found by VPREC, we run a mixed-precision version of the Newton-Raphson scheme where the first seven iterations use the `binary32` format. We note that the convergence speed and final result are almost identical to the full `binary64`

version. While more efficient, the solution found by VPREC has only been validated for a given input data set and its resiliency has to be examined as discussed in section 4.2.

3 Exploring variable precision

In large computational scientific codes, basic numerical computation functions are used many times in various places. Each call of the function may require a different numerical precision. Furthermore, as seen in section 2, the same code section may require a different precision over time. To explore the temporal dimension of numerical precision, we use two components presented in the remaining of this section:

- A variable precision backend for Verificarlo: VPREC, detailed in section 3.1;
- A heuristic optimizer to automatically minimize the precision configuration, while ensuring accuracy and convergence as described in section 3.2.

3.1 Verificarlo and the VPREC backend

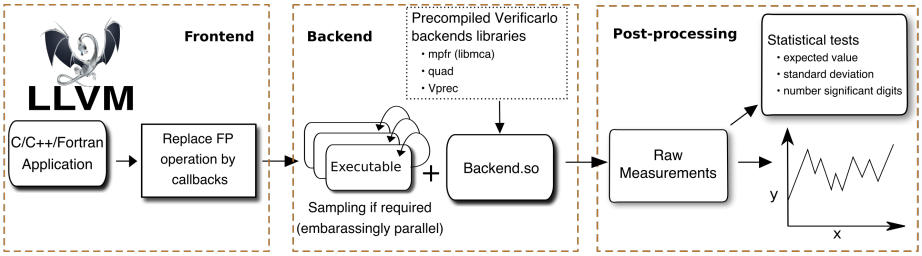

Fig. 2. General Verificarlo exploration workflow with VPREC backend

Verificarlo [14] is an open-source tool for FP interposition whose workflow is described in figure 2. The LLVM front-end replaces every FP operations by a call to the Verificarlo interface. After compilation, the program is dynamically linked against various backends [7, 14]. Interposing FP operations at the compiler level allows to capture the compiler optimization effects on the generated FP operation flow. Furthermore, it reduces the interposition overhead by optimizing its integration with the original code.

The VPREC backend simulates FP formats that can fit into the IEEE-754 double precision format, avoiding the complex engineering process to implement a shadow memory [36].

As illustrated in figure 3, the current implementation of VPREC allows modifying the bit length of the exponent $r\ 2\ [1, 11]$ and the pseudo-mantissa

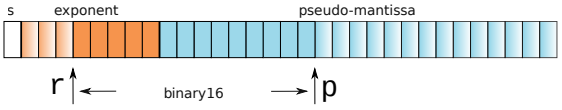

Fig. 3. By setting $r = 5$ and $p = 10$, VPREC simulates a *binary*16 embedded inside a *binary*32. Opaque bits represent the new exponent range and precision available. The sign remains the same.

p *2* $[0, 52]$. The explored format can be converted back and forth to double without loss of precision. At each instrumented floating-point operation, VPREC rounds the operands in (r, p), converts them to double, performs the operation in double precision, rounds the result using the *Round Ties to Even* mode, and stores it as a 64-bit number. This process presents two advantages:

- VPREC operands can be converted to double to use hardware operators to perform VPREC operations with low overhead, including special values (subnormal numbers, NaN and *1*) related to the user defined format (r, p)
- After rounding, converting the result back to double enables graceful degradation if some external libraries are not instrumented.

The user should note that in some rare cases, our implementation suffers from double rounding issue [5], when the first rounding done by the hardware at 53 bits impacts the second rounding to the required precision p. In practice, these cases, which occurs for numbers close to the midpoint of two p bits floating-point numbers, have no significant impact on our experiments.

The VPREC backend requires a single execution of the program and Verificarlo supports MPI. Therefore, we observe a reasonable overhead on full scale parallel applications ranging from 2.6 to 16.8 for very FP intensive codes.

3.2 Piecewise constant exploration heuristic

Let consider an iterative program with n iterations. VPREC simulates the effect of using FP numbers represented on a p_k-bit mantissa and r_k-bit exponent at iteration k. From now on, the sequence of values $[(p_0, r_0), \ldots, (p_n, r_n)]$ represents one VPREC run called a *con guration*.

When the precision is too low, the program execution fails: either it does not converge or it produces wrong results. We are only interested in *valid* configurations that preserve convergence and accuracy according to user knowledge. In our experiment (sec. 4), the validation function checks that the number of iterations and the final results are within a threshold of some reference values.

Reducing the precision can be seen as introducing numerical errors terms in the computation. Determining how and where to distribute the errors across the iterations is an optimization problem with a large number of valid configurations. Exploring the whole search-space is too costly on any non-trivial application, we have to rely on heuristics.

We propose the *piecewise* search heuristic with the three following design principles:

1. Configurations where the precision changes slowly over time are preferable to configurations which quickly oscillate between a low and high precision.
2. Precision lowering should be distributed among all iterations.
3. Early iterations are generally more robust to error. Therefore we foster configurations with lower precision in early iterations compared to late iterations.

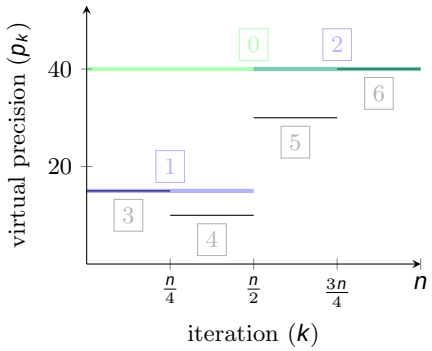

Fig. 4. First three steps of the piecewise constant search heuristic for the Newton-Raphson problem. The constant piecewise functions of step 0,1 and 2 are represented in green, blue and black.

To enforce the first and second principles, we follow a top-down approach. The solution is modeled by a piecewise constant function that is progressively refined. Figure 4 illustrates the first three steps of the piecewise approach on the Newton-Raphson problem. For the sake of simplicity, we are solely focusing on lowering the mantissa size p_k. However, the method can simultaneously deals with the mantissa and exponent size. Initially, the piecewise constant function has a single domain (marked 0) that spawns all the iterations $[0, n)$. The valid lowest precision, corresponding to 40, is found by dichotomy on the precision domain between 1 and 53. In the second step, the domain is split in two subdomains (marked 1 and 2). To enforce the third principle, the precision in the left domain (marked 1) is lowered in a greedy manner, while keeping the maximal precision found in the previous step for the right domain (marked 2). Once the lowest precision for domain 1 is identified, we lower the precision in domain 2. This ends the second step and produces the blue piecewise function. This process continues recursively. This approach guarantees by construction that the piecewise function at step i is an upper bound of the function at step $i + 1$ and progressively refines the solution following the first principle. The exploration is breadth-first to evenly distribute the reduction in precision, according to the second principle.

4 Large scale study validation on YALES2

YALES2 is a parallel CFD library that aims at solving the unsteady Navier-Stokes equations in the low-Mach number approximation for multiphase and reactive flows [28]. It efficiently handles unstructured meshes with several billions of elements, enabling the Direct Numerical Simulation of laboratory and industrial configurations. A projection method [8, 32] enforces the mass-conservation constraint on the flow thanks to the resolution of a Poisson equation at each time-step. In the HPC context, this is usually achieved with Krylov methods and we focus here on the Deflated Preconditioned Conjugate Gradient (DPCG) [29, 26, 27] implemented in YALES2. In this algorithm, a coarse grid is built from the fine mesh by merging a fixed number of elements together in super-cells (this procedure is conceptually similar to the multi-grid approach [13]). The general principle of deflation is the following. The coarse grid is used to converge the low frequency eigenmodes of the solution which represent the long range interactions of the Poisson equation. This requires much less work than performing a classical CG on the fine mesh which is only used to obtain the remaining high frequencies in a small number of iterations.

Numerically, the deflated operator is solved in $iterdef$ iterations using a usual PCG method such that the convergence criteria $convcrit$ is met. The solution is then expanded and injected into the main PCG loop on the finer grid. This whole process is repeated until the $maxnorm$ of the fine grid residual is below a threshold. In both CG solvers (on coarse and fine grids), the system is preconditioned by the inverse of the diagonal. In all experiments, we constraint the total number of iterations performed, $\sum iterdef_k$, by the algorithm to be below 1% of additional iterations compared to the original.

The representative use-case we focus on is the PRECCINSTA burner [24, 2]. It is a well-known lab-scale burner used to validate combustion CFD solvers. We use 3 different mesh sizes of 1.75 million, 40 million and 870 million of tetrahedral elements. For all configurations, the super-cell size in the coarse grid was set to 500 elements. We set the max norm of residual convergence criteria to 10^{-8}.

To reduce the search space, we consider two sets of functions. The first set, named *de ated*, is the set of functions used on the coarse grid to solve the deflated operator. The second set, named *all*, contained all the functions used to solve the fine grid operator.

4.1 Adaptive precision algorithm experiment on DPCG

In this section, we apply our VPREC tool on the 1.75M mesh case to explore valid variable precision implementation. In order to use true single precision, we statically set the exponent range to 8-bits in VPREC exploration.

Figure 5 shows the result of the exploration. In both graphics the x-axis represents the number of iterations on the fine grid. In the top plot, the right y-axis represents, on the same scale, the norm $maxnorm$ of the residual between two successive iterations and the convergence criterion $convcrit$ of the deflated operator. The left y-axis represents $iterdef$, the number of iterations on the

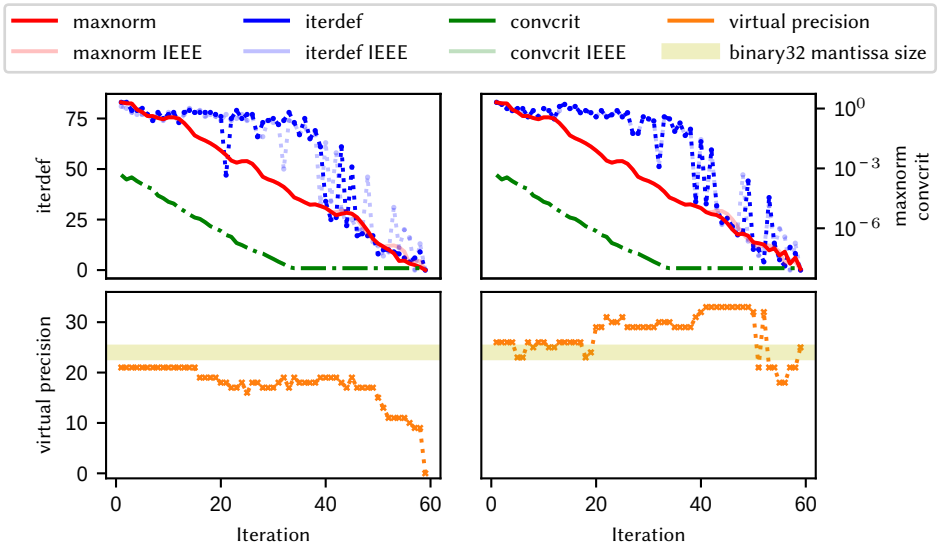

Fig. 5. Adaptive precision searching on YALES2's DPCG with the deflated part (*left*) and the entire code (*right*). On both plots, we can see that our reduced precision solution follows the reference IEEE convergence profile.

deflated operator. In the bottom plot, the y-axis represents the virtual precision used to compute the given iteration on x-axis.

Figure 5 *(left)* shows that less than 23 bits of precision are required for the deflated operator on the 1.75 million elements mesh, with an average precision of 18 bits. Therefore, the deflated operator can be computed with `binary32`, resulting in a mixed-precision implementation detailed in section 4.3.

Figure 5 *(right)* shows the results of the proposed explorations on both coarse and fine grid at the same time. We notice that, contrary to the deflated experiment, the required precision increases over iterations. This is expected because the solver needs more and more precision to converge as it refines the solution. Surprisingly, the required precision drops at iteration 50 from 34 bits to 21 bits. We cannot yet explain this sudden drop, more investigations are needed.

4.2 Validating resiliency to round-off errors

In the previous sections, we demonstrated that YALES2's DPCG converges with a lower precision format. This result is only valid with the particular rounding mode used by VPREC and is sensitive to the input dataset. In a realistic setup, small rounding errors may occur when performing FP operations with a different software representation and/or hardware.

Monte Carlo Arithmetic (MCA) is a stochastic method to model round-off errors by artificially introducing noise within computations and performing Monte

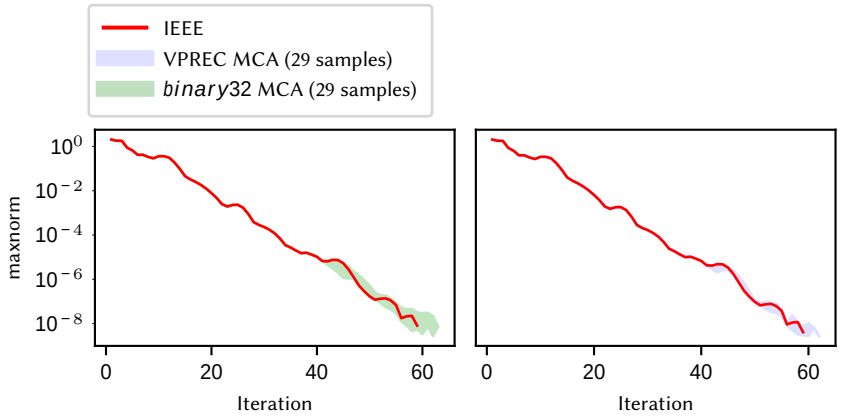

Fig. 6. Resiliency of VPREC and `binary32` configurations. In red the IEEE maxnorm convergence for reference. Blue envelop shows the 29 MCA samples for the previously found VPREC configuration. Green envelop shows the 29 MCA samples for the `binary32` configuration. All samples converge, showing the resiliency of both configurations.

Carlo sampling. For the theoretical underpinnings, readers may refer to [14, 31]. MCA is able to simulate rounding errors at a given virtual precision. We use MCA as a second step of our VPREC analysis to find a configuration that is resistant to round-off errors. We model this process as a Bernoulli trial. We run 29 MCA samples to simulate the effect of rounding errors. If any one of the samples fail to converge, we conclude that the solution is not robust to round-off errors. On the other hand, if all the 29 samples converge we conclude, thanks to the confidence intervals introduced in [37], that the probability of convergence in the presence of round-off errors is 90% with a 0.95 confidence level.

Figure 6 shows that the VPREC solution found in the previous section is robust and converges for all the samples. Since the solution is very close to the `binary32` precision, our objective is to achieve a robust `binary32` configuration. The binary32 constant-precision configuration represented by the light red envelop in figure 6 converges in 57 to 63 iterations in the presence of round-off errors for all samples. This demonstrates that it is possible to safely rewrite the coarse grid operator of DPCG in `binary32`.

4.3 Evaluating mixed-precision version

The deflated operator of DPCG can be computed within the `binary32` format for most iterations as shown in previous sections. To validate the results, we compiled a mixed-version of YALES2 where the deflated operator can be executed either in `binary32` or `binary64` format.

We evaluated the mixed precision version on the three different grids of PRECCINSTA and 10^{-9} convergence criteria. We limit the exploration algo-

rithm to double and single precision since we are not running on variable precision hardware.

We use CRIANN cluster constituted of 366 bisocket Intel Xeon E5-2680 nodes and Intel Omnipath interconnect. We gather statistics using Intel IPM interface for Intel MPI.

As predicted by our methodology, the computation converges and all versions satisfy all accuracy constraints on the results. However, we noticed that larger experiments require extra initial double precision iterations on the deflated grid. For examples, respectively two and four extra double precision iterations are necessary for the 40M and 870M mesh. This is coherent with the observations of Cools et al. [9] about the importance of being precise in the first iteration of a CG recurrence:

We noticed as well, that on these larger cases it is necessary to switch for the deflated grid from single to double precision when the deflated convergence criteria is difficult to reach with single precision 10^{-8}.

This effect did not appear on the smaller case with 500 elements per group. Our hypothesis is that the granularity difference between the two grids level is larger on the small mesh and therefore the small errors on the coarse grid iteration are less impacting on the fine grid iterations [26, 27].

We measure a 28% to 67% reduction in the communication volume. The energy gain can be estimated to be linearly related to this volume gain with the simple model proposed in [1].

Since DPCG is mostly bounded by communication latency, the performance gain is limited when the number of processor grows for a given size falling from 28% speedup to -2% slowdown on critical strong scaling experiments. However, according to these results and end-user usage of the code, the expected speedup for daily usage will be in the 10% range.

5 Related works and Background

Many tools and strategies have been developed for lowering precision in codes. For HPC purposes, the challenge is to have fast and scalable tools for addressing real world applications. A comparison with our methodology is presented in table 2. Most of the tools focus on the spatial dimension while we investigate the temporal dimension as well. In addition, most of them focus on the mixed-precision exploration while we provide a more in-depth analysis by working at a bit level. Evaluating the resiliency to rounding errors is only proposed by Verificarlo and Promise although FlexFloat [21] and fpPrecisionTuning [38] propose statistic optimization according to input data ranges. However, they require the re-implementation of code to adopt the specific libraries.

Daisy [10], Herbie [30] and STOKE [35] are optimizing precision or accuracy by rewriting formulas. Most of them provides high level of guarantees, however they all face scalability issues.

Some authors propose adaptive schemes for specific linear algebra algorithms. Anzt et al. [1, 19] propose an adaptive precision version of the Block-Jacobi

Tool	Localization	Mixed prec.	Variable prec.	Round. Error	Automatic
Precimonious [34]	Spatial	✓			✓
Blame-Analysis [33]	Spatial	✓	✓[1]		✓
Promise [16]	Spatial	✓		✓	✓
CRAFT [23]	Spatial	✓	✓[2]		✓
fpPrecisionTuning [21]	Spatial	✓	✓		
FlexFloat [38]	Spatial	✓	✓		
Verificarlo (this paper)	Temporal	✓	✓	✓	✓

Table 2. Comparisons of the different tools for exploring precision reduction.

preconditioner. Authors store data at low precision by truncating bits while computations remain in double precision. The change of format is guided by the condition number and the data range. The authors estimate energy gains with predictive models with the underlying hypothesis that the cost depend linearly on the bit length of the data. These methods are interesting because they use mathematical properties of numerical schemes for adapting precision over iterations. However, the authors focused on small program sections based on their knowledge at high engineering cost. Therefore, their results are restricted to a class of specific algorithms unlike our method which provides a broader exploration tool. Of course, educated developers are still required to take the final decision to use lower precision provided by our VPREC tool.

6 Conclusion

Reducing communication volume and computation cost is important to reach exascale computing. Tailoring the precision to the requirements of the application offers consequent savings in performance and energy. We presented a methodology to automatically and finely adapt the precision over time for numerical iterative schemes. Our methodology goes beyond mixed-precision approaches by exploring precision configurations at bit level. The method explores the precision requirements over time, and therefore chooses an optimal precision for each application phase. To guarantee the accuracy of the results, we validate the robustness of our solutions to rounding errors with the help of stochastic arithmetic. Finally, our experiments show that the methodology handles large HPC codes like the Combustion-CFD solver YALES2. For YALES2, our approach shows that lowering the precision is viable and achieves 28% to 67% reduction in the communication volume, lowering the energy and runtime cost.

Acknowledgments We thank Exascale Computing Research Lab supported by CEA, Intel, and UVSQ. This work has been granted access to the HPC resources of CINES under the allocation 20XX-A0031010295 made by GENCI and the computing resources of CRIANN (Normandy, France).

References

1. Anzt, H., Dongarra, J., et al.: Adaptive precision in block-jacobi preconditioning for iterative sparse linear system solvers. Concurrency and Computation: Practice and Experience p. e4460 (2017)
2. Benard, P., Lartigue, G., et al.: Large-eddy simulation of the lean-premixed preccinsta burner with wall heat loss. Proceedings of the Combustion Institute (2018)
3. Benard, P., Viré, A., et al.: Large-eddy simulation of wind turbines wakes including geometrical effects. Computers Fluids **173**, 133 – 139 (2018). https://doi.org/https://doi.org/10.1016/j.compfluid.2018.03.015, http://www.sciencedirect.com/science/article/pii/S0045793018301154
4. Benard, P., Balarac, G., et al.: Mesh adaptation for large-eddy simulations in complex geometries. International Journal for Numerical Methods in Fluids **81**(12), 719–740 (2016). https://doi.org/10.1002/fld.4204, https://onlinelibrary.wiley.com/doi/abs/10.1002/fld.4204
5. Boldo, S., Melquiond, G.: When double rounding is odd. In: 17th IMACS World Congress. p. 11. Paris, France (2005)
6. Boulet, L., Bénard, P., et al.: Modeling of conjugate heat transfer in a kerosene/air spray flame used for aeronautical fire resistance tests. Flow, Turbulence and Combustion **101**(2), 579–602 (Sep 2018). https://doi.org/10.1007/s10494-018-9965-8, https://doi.org/10.1007/s10494-018-9965-8
7. Chatelain, Y., de Oliveira Castro, P., et al.: Veritracer: Context-enriched tracer for floating-point arithmetic analysis. In: 25th IEEE Symposium on Computer Arithmetic, (ARITH). pp. 61–68 (2018)
8. Chorin, A.J.: Numerical solution of the navier-stokes equations. Mathematics of computation **22**(104), 745–762 (1968)
9. Cools, S., Yetkin, E.F., et al.: Analysis of rounding error accumulation in Conjugate Gradients to improve the maximal attainable accuracy of pipelined CG. Research Report RR-8849, Inria Bordeaux Sud-Ouest (Jan 2016), https://hal.inria.fr/hal-01262716
10. Darulova, E., Horn, E., Sharma, S.: Sound mixed-precision optimization with rewriting. In: Proceedings of the 9th ACM/IEEE International Conference on Cyber-Physical Systems. pp. 208–219. IEEE Press (2018)
11. Das, D., Mellempudi, N., et al.: Mixed precision training of convolutional neural networks using integer operations. CoRR **abs/1802.00930** (2018), http://arxiv.org/abs/1802.00930
12. Defour, D.: Fp-anr: A representation format to handle floating-point cancellation at run-time. In: 25th IEEE Symposium on Computer Arithmetic, (ARITH). pp. 76–83 (2018)
13. Dendy, J.: Black box multigrid. Journal of Computational Physics **48**(3), 366 – 386 (1982). https://doi.org/https://doi.org/10.1016/0021-9991(82)90057-2, http://www.sciencedirect.com/science/article/pii/0021999182900572
14. Denis, C., de Oliveira Castro, P., Petit, E.: Verificarlo: Checking floating point accuracy through monte carlo arithmetic. In: 23nd IEEE Symposium on Computer Arithmetic, (ARITH). pp. 55–62 (2016)
15. de Dinechin, F., Pasca, B.: Designing custom arithmetic data paths with FloPoCo. IEEE Design & Test of Computers pp. 18–27 (2011)
16. Graillat, S., Jézéquel, F., et al.: Promise: floating-point precision tuning with stochastic arithmetic. In: Proceedings of the 17th International Symposium on Scientific Computing, Computer Arithmetics and Verified Numerics (SCAN). pp. 98–99 (2016)

17. Guedot, L., Lartigue, G., Moureau, V.: Design of implicit high-order filters on unstructured grids for the identification of large-scale features in large-eddy simulation and application to a swirl burner. Physics of Fluids **27**(4), 045107 (2015). https://doi.org/10.1063/1.4917280
18. Gustafson, Yonemoto: Beating floating point at its own game: Posit arithmetic. Supercomput. Front. Innov.: Int. J. **4**(2), 71–86 (2017)
19. Haidar, A., Tomov, S., et al.: Harnessing gpu tensor cores for fast fp16 arithmetic to speed up mixed-precision iterative refinement solvers. In: Proceedings of the International Conference for High Performance Computing, Networking, Storage, and Analysis. pp. 47:1–47:11. SC '18, IEEE Press, Piscataway, NJ, USA (2018)
20. Higham, N.J.: Accuracy and stability of numerical algorithms. Siam (2002)
21. Ho, N.M., Manogaran, E., et al.: Efficient floating point precision tuning for approximate computing. In: Design Automation Conference (ASP-DAC), 22nd Asia and South Pacific. pp. 63–68. IEEE (2017)
22. Intel Corp.: Intel vml (2018), https://software.intel.com/en-us/mkl-developer-reference-c-vector-mathematical-functions
23. Lam, M.O., Hollingsworth, J.K., et al.: Automatically adapting programs for mixed-precision floating-point computation. In: Proc. of the 27th International conference on supercomputing. pp. 369–378. ACM (2013)
24. Lartigue, G., Meier, U., Bérat, C.: Experimental and numerical investigation of self-excited combustion oscillations in a scaled gas turbine combustor. Applied thermal engineering **24**(11-12), 1583–1592 (2004)
25. Legrand, N., Lartigue, G., Moureau, V.: A multi-grid framework for the extraction of large-scale vortices in large-eddy simulation. Journal of Computational Physics **349**, 528 – 560 (2017). https://doi.org/https://doi.org/10.1016/j.jcp.2017.08.030, http://www.sciencedirect.com/science/article/pii/S0021999117306010
26. Malandain, M.: Massively parallel simulation of low-Mach number turbulent flows. Theses, INSA de Rouen (Jan 2013), https://tel.archives-ouvertes.fr/tel-00801502
27. Malandain, M., Maheu, N., Moureau, V.: Optimization of the deflated conjugate gradient algorithm for the solving of elliptic equations on massively parallel machines. Journal of Computational Physics **238**, 32 – 47 (2013). https://doi.org/https://doi.org/10.1016/j.jcp.2012.11.046, http://www.sciencedirect.com/science/article/pii/S0021999112007280
28. Moureau, V., Domingo, P., Vervisch, L.: Design of a massively parallel CFD code for complex geometries. Comptes Rendus Mécanique (339), 141–148 (2011)
29. Nicolaides, R.A.: Deflation of conjugate gradients with applications to boundary value problems. SIAM Journal on Numerical Analysis **24**(2), 355–365 (1987)
30. Panchekha, P., Sanchez-Stern, A., et al.: Automatically improving accuracy for floating point expressions. In: Proceedings of the 36th ACM SIGPLAN Conference on Programming Language Design and Implementation. pp. 1–11. ACM (2015)
31. Parker, S.: Monte carlo arithmetic: exploiting randomness in floating-point arithmetic. Tech. Rep. CSD-970002, UCLA Computer Science Dept. (1997)
32. Pierce, C.D., Moin, P.: Progress-variable approach for large-eddy simulation of non-premixed turbulent combustion. Journal of Fluid Mechanics **504**, 73–97 (2004). https://doi.org/10.1017/S0022112004008213
33. Rubio-González, C., Nguyen, C., et al.: Floating-point precision tuning using blame analysis. In: Proceedings of the 38th International Conference on Software Engineering. pp. 1074–1085. ACM (2016)
34. Rubio-González, C., Nguyen, C., et al.: Precimonious: Tuning assistant for floating-point precision. In: International Conference for High Performance Computing, Networking, Storage and Analysis (SC). pp. 1–12. IEEE (2013)

35. Schkufza, E., Sharma, R., Aiken, A.: Stochastic optimization of floating-point programs with tunable precision. ACM SIGPLAN Notices **49**(6), 53–64 (2014)
36. Serebryany, K., Bruening, D., et al.: Addresssanitizer: A fast address sanity checker. In: USENIX ATC 2012 (2012)
37. Sohier, D., De Oliveira Castro, P., et al.: Confidence Intervals for Stochastic Arithmetic (2018), https://hal.archives-ouvertes.fr/hal-01827319, preprint
38. Tagliavini, G., Mach, S., et al.: A transprecision floating-point platform for ultra-low power computing. In: Design, Automation & Test in Europe Conference & Exhibition (DATE). pp. 1051–1056. IEEE (2018)