

Verificarlo CI: Continuous Integration for Numerical Optimization and Debugging

Aurélien Delval^{a,b,*}, François Coppens^a, Eric Petit^c, Roman Iakymchuk^d and Pablo de Oliveira Castro^a

^aUniversité Paris-Saclay, UVSQ, LI-PaRAD, 9 boulevard d'Alembert, bâtiment François Rabelais, 78280 Guyancourt, France

^bSiPearl, 44 rue Jean Mermoz, 78600 Maisons-Laffitte, France

^cIntel Corporation, 2200 Mission College Blvd., M/S RNB4-145, Santa Clara, 95054, CA, USA

^dUmeå University and Uppsala University, 90187 Umeå, Sweden

ARTICLE INFO[†]

Keywords:

Continuous Integration / Continuous
Deployment;
Verificarlo;
Numerical Accuracy;
High-Performance Computing

ABSTRACT

Floating-point accuracy is an important concern when developing numerical simulations or other compute-intensive codes. Tracking the introduction of numerical regression is often delayed until it provokes unexpected bug for the end-user. In this paper, we introduce Verificarlo CI, a continuous integration workflow for the numerical optimization and debugging of a code over the course of its development. We demonstrate applicability of Verificarlo CI on two test-case applications.

1. Introduction

Despite Floating-point (FP) accuracy being a known issue [1], modern tools for software development do not provide automated numerical accuracy regression tests. To fill this need, we propose Verificarlo CI (Continuous Integration). GitHub and GitLab are popular platforms for developing software, and both have features for CI. CI services are triggered on specific events, such as merging a pull request. Verificarlo CI is designed to be integrated with them, but it can also be used with custom workflows.

To facilitate its adoption, Verificarlo CI has been designed to be easy and fast to deploy, while still being flexible enough to be relevant for most applications. We provide the user with a simple API to insert FP probes in their tests, execute them with Verificarlo, setup CI Actions, and finally access and interpret the results.

Finally, we demonstrate Verificarlo CI on two use-cases: exploring reduced mixed-precision in the Nekbone CFD proxy application; tracking numerical bugs during the development of the Quantum Monte Carlo Chemistry Kernel library (QMCKl).

2. Verificarlo CI for numerical correctness

Verificarlo [2] is a tool based on the LLVM compiler framework modifying at compilation each floating point operation with custom operators. After compilation, the program can be linked against various backends to explore FP related issues and optimizations. The latest version of Verificarlo fully supports OpenMP and MPI parallel programs.


Verificarlo computes the number of significant bits to evaluate the numerical accuracy of a computation. It captures the number of accurate bits in the FP mantissa against a chosen reference. Unfortunately, an exact reference value is not known beforehand for many complex programs or intermediate computations. To overcome this problem, Verificarlo uses Monte Carlo arithmetic (MCA) [3], a stochastic method that can simulate numerical errors and estimate the number of significant bits directly: the result of each FP operation is replaced by a perturbed computation modeling the losses of accuracy. From a set of MCA samples, it is possible to estimate the significant bits of a computation, $s_2 = -\log_2 |\sigma/\mu|$, where σ and μ are the sample's standard deviation and mean [4].

Verificarlo includes six backends, which are extensively documented in the user manual. The two most important backends are: the MCA backend, described previously, and the VPREC backend that emulates the effect of using mixed-precision in a program [5].

Verificarlo CI automates numerical accuracy tests by using a separate Git branch to store test results. Whenever users make modifications to the main branch, a remote runner carries out predefined tests and uploads the results

[†]This paper is part of the ParCFD 2024 Proceedings. A recording of the presentation is available on YouTube. The DOI of this document is 10.34734/FZJ-2025-02465 and of the Proceedings 10.34734/FZJ-2025-02175.

*Corresponding author

 aurelien.delval@uvsq.fr (A. Delval); francois.coppens@uvsq.fr (F. Coppens); eric.petit@intel.com (E. Petit); riakymch@cs.umu.se (R. Iakymchuk); pablo.oliveira@uvsq.fr (P. de Oliveira Castro)

ORCID(s): 0000-0002-2707-6940 (A. Delval); 0000-0003-1695-352X (F. Coppens); - (E. Petit); 0000-0003-2414-700X (R. Iakymchuk); 0000-0001-9007-6145 (P. de Oliveira Castro)

to the CI branch. Users can view their results dynamically using a simple web server.

Verificarlo CI offers a command-line interface that helps configuring the CI pipeline on a given application and hooks it up to a GitLab or GitHub repository : it automatizes the initial setup by creating both the CI pipeline and the dedicated results branch. Users are then free to further customize their pipelines.

Developers use a dedicated C or Fortran API to include *probes* in their code. Each probe is associated with a test and a variable name. During testing, the probe measures the accuracy of a chosen variable. Optionally, an alert can be triggered if the relative or absolute error exceeds a user-set threshold. In the below C example, the probe is identified by the "test"/"var" couple, and an absolute error threshold is set to 0.01:

```
vfc_probe_check(probes, "test", "var", var, 0.01);
```

In order to be able to run the tests, Verificarlo CI requires a description of the tests and backends to run. It is specified in a JSON configuration file which supports complex test setups. The test results are exported to an HDF5 file, a hierarchical format commonly used in HPC applications. Test results are stored on the dedicated CI branch, allowing robust archival. The HDF5 files can optionally embed the raw test results. In the default CI workflow, this raw data is stored as a job artifact and accessible for a limited time, to enable user defined additional analysis. Finally, Verificarlo CI analyzes the data and generates dynamic reports organized into different views: temporal, cross-test, or cross-variable comparisons and accuracy violations.

3. Mixed-precision for Nekbone proxy application

Nek5000¹ is a high-order solver for Computational Fluid Dynamics (CFD) based on the Spectral Element Method (SEM) that solves the Navier-Stokes equation for incompressible flow. Nekbone is a proxy application for Nek5000 that focuses on important computational and scaling aspects of the entire solver. We used the VPREC backend to examine precision appetites in Nekbone using the polynomial degree of 10 and different number of elements. The results of tracking the residual of the Conjugate Gradient (CG) solver, see Fig. 1, suggest a possibility of using as little as 16 bits of mantissa (the beginning of the plateau) and still being able to converge, while the original version relies on FP64 (double precision) with 52 bits of mantissa. We verified this assumption with the MCA backend, confirming such a possibility for the precision reduction to single in the CG solver on CPUs.

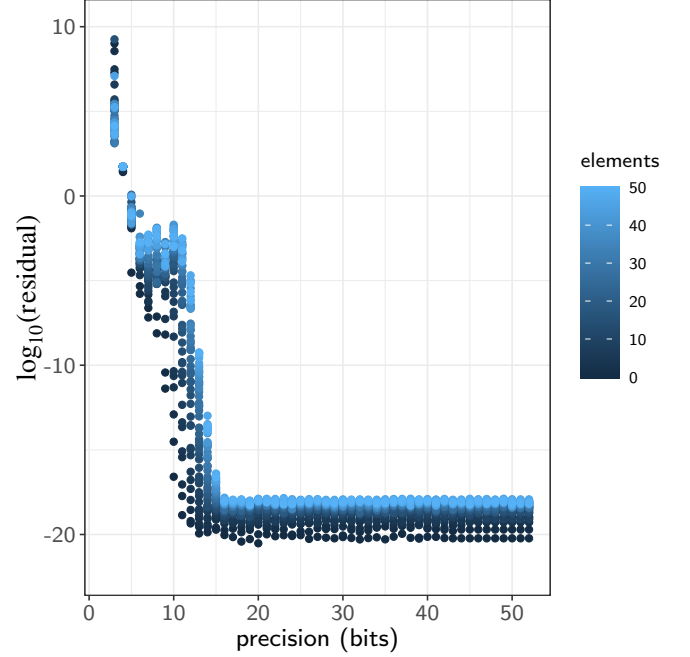


Figure 1: Examining precision needs in Nekbone for various numbers of elements: the residual (L^2 norm) in CG.

Recently, following this precision inspection, we modified Nekbone to support mixed single-double precision and we were able to reduce the time-to-solution by 34 %. Once a suitable precision is found, a Verificarlo CI probe can be inserted in the code, to automatically monitor the residual error of each subsequent code version.

4. Tracking accuracy in the QMckl library

The Sherman-Morrison-Woodbury (SMWB) kernel was developed as part of the QMckl library², an open-source library of highly-optimized building blocks for implementing Quantum Monte Carlo methods in the TREX European Center of Excellence.

Given a matrix A and its inverse A^{-1} , Sherman-Morrison (SM) is a formula to efficiently compute the inverse after a rank-1 update uv^T on A

$$(A + uv^T)^{-1} = A^{-1} - \frac{A^{-1}uv^T A^{-1}}{1 + v^T A^{-1}u}. \quad (1)$$

This formula can be generalized for rank- k updates using the Woodbury (WB) formulation [6]. In WB, the denominator of SM is replaced by the inverse of a small square $k \times k$ matrix. For $k = 2$ and $k = 3$, WB is expected to be faster than

¹<https://nek5000.mcs.anl.gov/> and <https://github.com/Nek5000/Nekbone>

²<https://github.com/TREX-CoE/qmckl>

iterating 2 or 3 times with SM. QMCKl implements different algorithms to apply SM with a set of updates (u_j, v_j) , for $j = 1, \dots, m$. The naive approach, *SM1*, applies these updates in sequence.

Depending on the updates ordering, the SM denominator can be close to zero, meaning that the matrix A becomes singular. This makes the method numerically unstable. A refined algorithm using Slagel splitting [7] is called *SM2*. Below a minimum threshold for the denominator, the update is split in two, the first half is applied, and the second half enqueued with remaining updates.

To implement the Woodbury formula, blocks of rank-3 and rank-2 updates are built. If the intermediate matrix update is singular, the corresponding updates are split with *SM2*. This method is called *SMWB*. Since *SMWB* changes the order of operations, one must ensure that the numerical accuracy is preserved compared to *SM2*.

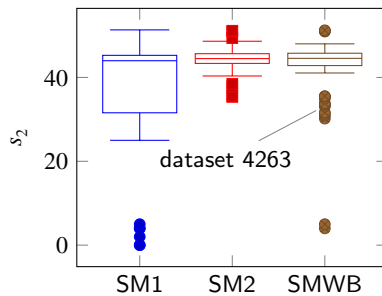


Figure 2: Significant bits of Frobenius norm, for all datasets and algorithm combinations, for commit 6f282, grouped by algorithms. SMWB fails catastrophically in some cases.

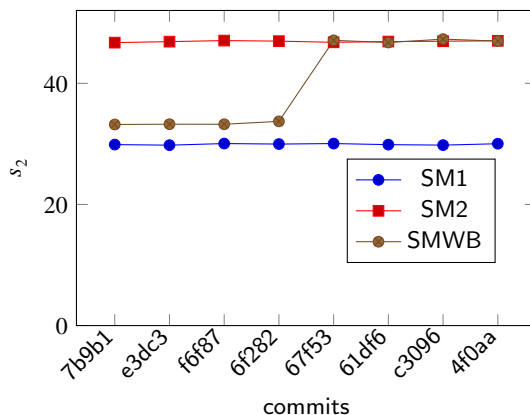


Figure 3: Significant bits of Frobenius norm, for our different algorithms, over commits for dataset 4263. SMWB’s accuracy improves after the fix.

To track the accuracy of these algorithms during development with Verificarlo CI, we use a large number of datasets from a QMC=Chem [8] use case on Benzene. All datasets are run with all algorithms in MCA mode. The main development branch in the repository was instrumented with probes identified with *dataset number / algorithm* couples allowing a factored analysis in the dynamic reports. Finally, we set up a CI branch using the command-line helper to track accuracy on the main development branch, from which Fig. 2 and Fig. 3 were generated.

During the development of SMWB, the *run inspection* report, reproduced in Fig. 2, highlighted some outputs for which SMWB fails with a high error under the MCA backend. After investigation, we discovered that in the initial implementation of SMWB, delayed updates were directly applied after each WB step. This reduces the numerical stability because it increases the probability of producing singular intermediate matrices. It was fixed in commit 67f53 by moving all the updates to the very end of the update queue as shown in Fig. 3, obtained from the *temporal view*.

5. Conclusion

Verificarlo CI automates numerical accuracy tests within a continuous integration workflow: it grants users the ability to define such tests. It provides an easy way to visualize results throughout the development process of a code. Better integration of numerical checks in the CI process saves developers precious time to focus on their area of expertise.

Verificarlo CI has been used in the context of TREX and CEEC EuroHPC JU Centers of Excellence to detect numerical regressions, pin-point faulty commits, and predict the effect of mixed-precision. A tutorial demonstrating its use is available on GitHub³. Furthermore, we believe that using such a tool as a part of the regular CI/ CD process would help for early stage identification of numerical bugs and re-ensuring numerical reliability of codes.

Acknowledgements

This work was partially supported from EC by EuroHPC Centers of Excellence TREX (952165) and CEEC (101093393), as well as by the French National Agency for Research via the InterFLOP project (ANR-20-CE46-0009).

References

- [1] D. Goldberg, What every computer scientist should know about floating-point arithmetic, *ACM Computing Surveys* 23 (1) (1991) 5–48. doi:10.1145/103162.103163.
- [2] C. Denis, P. De Oliveira Castro, E. Petit, Verificarlo: Checking Floating Point Accuracy through Monte Carlo Arithmetic,

³https://github.com/verificarlo/vfc_ci_tutorial

- in: 2016 IEEE 23rd Symposium on Computer Arithmetic (ARITH), IEEE, 2016, pp. 55–62. doi:10.1109/ARITH.2016.31.
- [3] D. S. Parker, D. Langley, Monte carlo arithmetic: exploiting randomness in floating-point arithmetic (1997).
URL <https://api.semanticscholar.org/CorpusID:2321215>
 - [4] D. Sohier, P. D. O. Castro, F. F  votte, B. Lathuili  re, E. Petit, O. Jamond, Confidence Intervals for Stochastic Arithmetic, ACM Transactions on Mathematical Software 47 (2) (2021) 1–33. doi:10.1145/3432184.
 - [5] Y. Chatelain, E. Petit, P. de Oliveira Castro, G. Lartigue, D. Defour, Automatic Exploration of Reduced Floating-Point Representations in Iterative Methods, Vol. 11725, 2019, pp. 481–494. doi:10.1007/978-3-030-29400-7_34.
 - [6] M. Woodbury, P. U. D. of Statistics, Inverting Modified Matrices, Memorandum Report / Statistical Research Group, Princeton, Department of Statistics, Princeton University, 1950.
URL https://books.google.de/books?id=_zAnzgEACAAJ
 - [7] J. T. Slagel, The sherman morrison iteration, Master’s thesis, Virginia Tech (2015).
 - [8] A. Scemama, M. Caffarel, E. Oseret, W. Jalby, QMC=Chem: A Quantum Monte Carlo Program for Large-Scale Simulations in Chemistry at the Petascale Level and beyond, Vol. 7851, 2013, pp. 118–127. doi:10.1007/978-3-642-38718-0_14.