Debugging and Optimization of HPC Programs in Mixed Precision with the Verrou Tool

Index Terms—Floating-point arithmetic, Verrou, mixed precision, Verification & Validation

I. INTRODUCTION

Floating-Point (FP) arithmetic is becoming a hotter and hotter topic in High-Performance Computing (HPC). First, high computational performance is often achieved at the expense of a loss of control over the order in which FP operations are executed: vectorization, aggressive compiler optimizations and the use of parallel clusters are only a few examples why the execution flow of a HPC code might change. Since FP arithmetic is not associative, this change in the order of operations leads to a loss of the reproducibility of the results, which can in turn have a wide array of consequences, ranging from benign difficulties in the debugging process, to more serious issues with the Verification & Validation (V&V) process.

Second, optimizing the use of FP precision is often key to achieving high performance. Using smaller FP numbers allows reducing the memory bandwidth usage as well as increasing the number of simultaneous FP operations performed by a single SIMD instructions. Therefore, high FP precision should only be used where it is needed to ensure the required results quality, FP precision should be reduced everywhere possible.

In this paper, we present how the Verrou [1] tool can help deal with these issues in the context of large, industrial, high-performance scientific computing codes such as the ones developed and used by leading actors in the industry. In part II, the Verrou tool and its architecture will be briefly presented. Emphasis will be put on the recent development of new features specifically targetting mixed-precision codes. Part III will be devoted to more methodological topics and show how Verrou can be used, alongside with debugging tools, to debug large industrial scientific computing codes and help optimize their use of mixed precision. The full paper will contain an additional part in which these features will be demonstrated on a real industrial code.

II. PRESENTATION OF THE VERROU TOOL

Verrou is a tool aiming at helping diagnose, debug and optimize FP-related issues in large, industrial scientific computing codes. An earlier version of the Verrou tool has been presented in [1], where internal arithmetic-related algorithms were presented in detail. Only the most important aspects of Verrou will be recalled here, and this paper will rather focus on recent developments in Verrou. Specifically, the features related to the use of Verrou for the diagnosis of mixed precision programs will be detailed. Also, this paper will focus more on the presentation of the features themselves, and less on the internal implementation.

From a user standpoint, Verrou instruments the program (in its binary form), replacing each FP instruction in it with a variant implementing another type of FP arithmetic. Results of the computation are output like in any normal execution, except that they are affected by the cumulative effect of all perturbed FP instructions. Analyzing the observed change in the results allows estimating the global impact of FP arithmetic for the code.

In its recent versions, the architecture of Verrou follows this description, with 3 major components which will be described in more details in the remainder of this section.

- The front-end is the part of Verrou responsible for running the user-given program, replacing each FP instruction with a variant.
- The back-end implements variants of all FP instructions, effectively defining a new arithmetic to be used in place of the original one. Different back-ends can be used in Verrou, implementing various arithmetics.
- A post-processing stage is needed in order to analyze the effect resulting from the use of an alternate arithmetic. This is mostly left for the user to implement, but advice can be given as to what kind of post-processing should be performed.

This architecture of Verrou in three parts stems from the Interflop\(^1\) initiative, which aims at providing a uniform API allowing interoperability between front-ends and back-ends of several FP instrumentation tools such as Verrou and Verificalo [2].

A. Front-end

Verrou is based on Valgrind to perform a Dynamic Binary Instrumentation (DBI) of the given program, and replace each FP instruction by a variant implemented by the user-chosen back-end.

The main advantage of using Valgrind is the ease of use of Verrou for end-users: there is no need to manually change the program or even recompile it. All that is needed is to prefix the usual command-line with in order to invoke Valgrind with the Verrou tool:

```
valgrind --tool=verrou [VERROU_ARGS] \ PROGRAM [ARGS]
```

\(^1\)https://github.com/interflop/interflop
Valgrind – and therefore Verrou – is naturally compatible with MPI. Valgrind can also instrument multi-threaded applications, but in this case threads will be “sequentialized”, which incurs a large performance overhead.

In the command above \texttt{VERROU_ARGS} allows changing several aspects of Verrou, particularly the arithmetic back-end to be used (described in the next paragraph). It is also worth noting that the Verrou front-end can be instructed to only instrument part of a program. This feature is key to many debugging and optimization methodologies described in section III.

### B. Back-ends

Several back-ends in Verrou define variants of FP arithmetic to be used in place of the standard FP arithmetic. The historical Verrou backend implements a stochastic arithmetic, in which the result of each FP instruction is randomly rounded either upward or downward. Depending on the chosen probability law, this can be similar to an asynchronous CESTAC arithmetic [3] or a variant of Monte-Carlo Arithmetic [4].

The only recent changes to this back-ends are related to mixed-precision programs: type conversions subject to rounding (e.g. \texttt{double} \rightarrow \texttt{float}) are now instrumented and perturbed when applicable.

More recently, a reduced-precision back-end has been introduced in Verrou. This back-end emulates the use of an IEEE-754-compliant FP arithmetic with reduced precision. It is currently implemented in only one variant, which reduces double precision FP numbers to single precision. This is similar to what is performed in the rpe library [5].

### C. Post-processing

Due to the changes introduced by the arithmetic emulated by Verrou, each execution of a given program (with a given dataset) produces perturbed results. An analysis of these results should be performed in order to assess the impact of FP arithmetic on the results quality.

Such post-processing is highly dependent on the nature and format of the results and, as such, it is mostly left for the user to implement for their own code. However in most instances, almost no additional work is needed when a Quality Assurance process with automated non-regression testing has been put in place. For example, it is possible to simply check whether the non-regression test suite succeeds when running the program within Verrou with the reduced-precision arithmetic back-end. If so, this gives a good indication that the program can be changed to use single-precision FP arithmetic while remaining “valid”.

Post-processing can get more complicated in the case of stochastic arithmetic, where several sampled results are needed to compute statistics. This aspect is detailed in [6].

### III. DEBUGGING AND OPTIMIZING HPC CODES WITH VERROU

The features of Verrou which have been presented above can be leveraged to set up methodologies aiming at debugging FP-related issues in industrial codes. An example is described in [7], which presents the complete analysis and debugging of code_aster [8], a structural mechanics simulation tool of more than 1.2M lines of code. This study benefits from the strong Quality Assurance process of code_aster, comprising around 4 000 test cases, of which around 2 000 are run daily.

The two cornerstones of this debugging methodology are:

1) the detection of unstable tests (i.e. for which the branch taken depends on previous round-off errors), based on the comparison between perturbed and unperturbed code coverages;

2) the detection of unstable functions or source code lines (i.e. parts of the code which, when perturbed, produce large perturbations in the results). This detection leverages the Delta-Debugging [9] algorithm to perform a binary search in the source code; it heavily relies on the Verrou ability to restrict the scope of FP perturbations to a part of the source code.

Recent improvements have been made to the Delta-Debugging algorithm in Verrou, in order to make it more parallelizable.

In this paper, we focus on the use of these debugging techniques in order to optimize mixed-precision implementations of a given code. Indeed, two complementary workflows can be devised, which help developers decide where high FP precision is needed.

First, in the case of single-precision codes, the debugging techniques described above can be used with the stochastic arithmetic back-end of Verrou. Issues found during the process can be fixed by promoting the relevant variables to double precision.

Second, in the case of double-precision codes, the same debugging techniques can be used with the reduced-precision back-end of Verrou. This time, issues found during the process point at variables which have to stay in double precision.

### IV. CONCLUSIONS

Various new features have been recently implemented in the Verrou tool, which allow performing the complete analysis of FP-related issues in mixed precision, high-performance codes. Most notably, existing back-ends have been adapted to correctly handle mixed precision, and a new backend has been introduced to emulate it. The delta-debugging-based localization of the origin of FP-related issues in the source code can now be performed in parallel.

These new features make it possible to use Verrou not only as a diagnostics tool, but also as an help during the mixed-precision optimization process.

### REFERENCES


