Evaluating the Performance Impact of Floating-Point Type Demotion in a Multiphase Flow Simulator

Vitor Barros Aquino[1], Pedro Henrique Casarotto Rigon[2], Gabriel Freytag[2], Afranio Jose de Melo Junior[3], Eduardo Ferreira Gaspari[3], Evaldo Costa[1]

[1]CESAR School
Recife - PE – Brazil

[2]CESAR - Centro de Estudos e Sistemas Avançados do Recife
Recife - PE – Brazil

[3]Petróleo Brasileiro S.A.
Rio de Janeiro – RJ — Brasil

vba@cesar.school, {phcr, gf2}@cesar.org.br

{afranio.melo, eduardogaspari}@petrobras.com.br, ebc@cesar.school

Abstract. *This paper evaluates precision demotion in Marlim3, an open-source 1D multiphase flow simulator developed by Petrobras. We replace* `long double` *declarations, extended-precision literals, and long-double* `libm` *calls with* `double`*-precision equivalents. The original and modified versions are compared using nine simulation cases on x86-64 and ARM platforms, considering runtime and numerical differences in the outputs. On x86-64, the modified version improved all workloads, reaching up to 11.69× speedup. On Apple M4, most workloads remained close to the baseline. Numerical validation reused the baseline workflow: steady-state network outputs match the extended-precision reference within engineering tolerance, while transient cases show only the expected accumulation of round-off, with all monitored quantities physically plausible. These results show that precision demotion can improve Marlim3 performance, but its impact is strongly architecture-dependent.*

1. Introduction

Scientific simulation codes often carry legacy implementation choices that become performance bottlenecks. In Marlim3, an open-source 1D multiphase flow simulator, we identified `long double` declarations, extended-precision literals, and long-double `libm` calls (e.g., `powl`, `sqrtl`). While useful for numerical robustness, these constructs introduce unnecessary overhead when double precision suffices.

This paper evaluates a source-level precision demotion from `long double` to `double` in Marlim3, transforming data types, literals, and mathematical functions. We compare the original and modified versions across nine representative workloads on x86-64 and Apple M4 platforms, focusing on the runtime-accuracy trade-off. The transformation is highly beneficial on x86-64 (up to 11.69× speedup) and mostly neutral on the M4, with all workloads meeting the baseline validation criteria (Section 3.4). We characterize the legacy use of extended-precision constructs in Marlim3 and provide a reusable source-level transformation workflow.

2. Background and Motivation

2.1. Marlim3

Marlim3 is an open-source 1D multiphase flow simulator developed by Petrobras. It supports steady-state and transient analyses of production and injection systems, including wells, flow networks, gas lift loops, compositional fluids, natural convection, and thermal diffusion coupled to the 1D flow formulation [Petrobras 2026]. Its code base is mostly written in C++, with additional Fortran, C, and Python components, making it a useful case study for evaluating source-level optimizations in a real multiphase flow simulator.

2.2. Floating-Point Precision and Type Demotion

Floating-point computations are affected by representation, rounding, and evaluation order [Goldberg 1991]. Thus, changing precision may improve performance, but may also change numerical results. In scientific codes, higher precision is often used conservatively even when not all operations require it. Precision-tuning techniques such as Precimonious exploit this by searching for lower-precision declarations that satisfy accuracy constraints while reducing execution cost [Rubio-González et al. 2013]. Mixed-precision algorithms follow the same principle: different parts of a computation may use different precisions as long as the final result remains accurate enough for the application [Higham and Mary 2022]. In this work, we evaluate a source-level demotion from `long double` to `double`.

2.3. Optimization Opportunity in Marlim3

The opportunity studied here stems from extended-precision constructs that may be justified in numerically sensitive regions but increase execution cost when the extra precision is unnecessary. Since floating-point behavior also depends on compiler choices and target architecture [Monniaux 2008], we treat the demotion to `double` as an optimization requiring explicit validation, consistent with precision-tuning approaches where reduced precision is acceptable only when numerical requirements are still satisfied [Chiang et al. 2017].

3. Methodology

This section describes the precision demotion applied to Marlim3 and the protocol used to evaluate its impact on performance and numerical behavior.

3.1. Code Transformation

We applied a global source-level precision demotion to the C++ and Fortran parts of Marlim3. The transformation replaced `long double` types, extended-precision literals, and long-double mathematical functions with `double`-precision equivalents. No algorithmic changes were introduced, and output formatting and I/O routines were kept unchanged.

The transformation was performed with scripts and checked with text search tools. In the baseline version, we identified 3684 long-double `libm` calls (mostly `fabsl`, `powl`, `sqrtl`, `sinl`, and `cosl`); all were replaced by their double-precision counterparts, with no `long double` constructs remaining after the transformation.

Table 1. Simulation cases used in the evaluation.

Workload	Type
caso_centurion3	Transient
caso_centurion4	Transient
caso_centurion5	Transient
teste1_2zonas-2VGLS-2	Transient
teste1_parada_longo	Transient
rede_gasoduto	Network
rede_gasoduto2	Network
rede_gasoduto3	Network
rede_gasoduto4	Network

3.2. Experimental Setup

Both versions were compiled with the same CMake Release configuration, using `-O3`, `-march=native`, `-funroll-loops`, `-ffp-contract=off`, and OpenMP. The flag `-ffp-contract=off` disables floating-point contraction (implicit FMA), ensuring numerical consistency across compilers and optimization levels. Experiments were conducted on two architectures: an x86-64 platform with an AMD Ryzen 5 5600X (6C/12T, 32 MiB L3, 16 GiB DDR4-3600) using GCC 11.4.0 on Linux, and an Apple M4 (10 logical cores, macOS 15.7.4) using GCC 15. The platforms use different GCC versions; however, since the macOS ARM64 ABI already maps `long double` to 64-bit IEEE 754, the comparison isolates the architectural representation of the type rather than a compiler version.

3.3. Workloads

We evaluated nine Marlim3 cases covering transient and network simulations, two important execution modes of the simulator, as listed in Table 1. These span long-running transient scenarios (gas lift valves, natural convection) and complex gas pipeline network topologies.

3.4. Evaluation Protocol

Each workload was executed ten times per version. The minimum and maximum times were removed, and the remaining eight runs were averaged; 95% confidence intervals were computed via Student-t and are reported as error bars in Figure 1. Output directories were cleaned before each run. Performance was evaluated by comparing the cleaned mean runtime of the original and modified versions. Numerical validation reused the baseline workflow, comparing the modified outputs one-to-one against the extended-precision reference. Steady-state network quantities matched within engineering tolerance; transient cases exhibited round-off differences that grow along the simulation horizon — an expected behavior of nonlinear dynamical systems — while all monitored quantities remained physically plausible, and PVT/flash properties and scheduled events were identical between versions.

4. Experimental Results

Figure 1 compares the execution time of the original and optimized versions. On x86-64, the optimized version reduced runtime in all workloads, with the largest absolute reductions in the longer transient cases (`caso_centurion5`, `teste1_2zonas-2VGLS-2`,

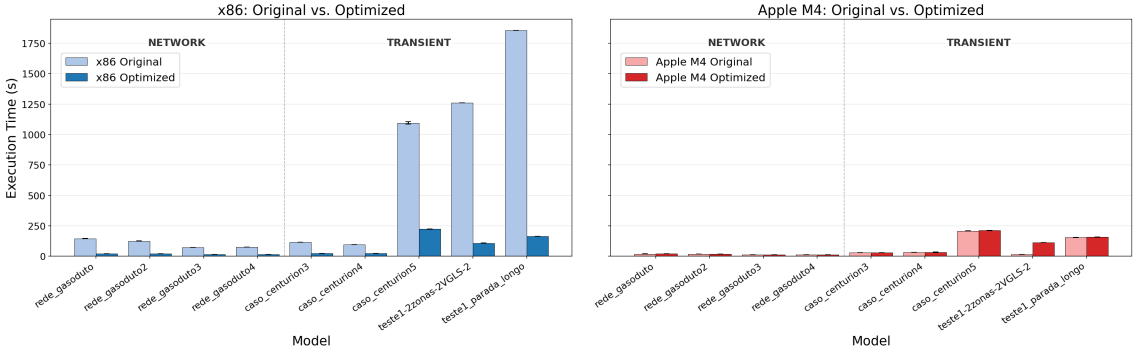

Figure 1. Clean mean execution time of the original and optimized versions. Error bars denote 95% confidence intervals.

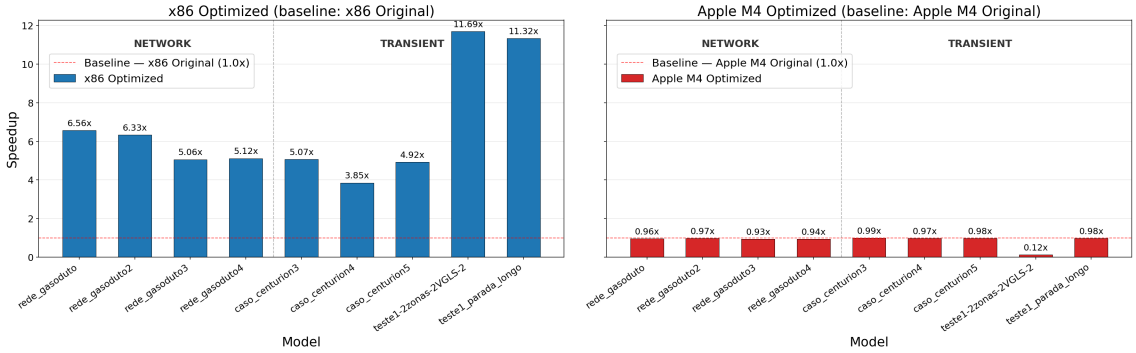

Figure 2. Speedup of the optimized version relative to the original (values >1.0 indicate improvement).

and `teste1_parada_longo`), which exercise the demoted code paths most intensively. On Apple M4, most workloads stayed close to the original time, confirming that the benefit depends on the architectural representation of extended precision.

Figure 2 summarizes the speedup obtained by the optimized version. On x86-64, all workloads improved, from 3.85× (`caso_centurion4`) to 11.69× (`teste1_2zonas-2VGLS-2`), with network cases between 5.06× and 6.56×. On Apple M4, the modified version stayed at or marginally below the baseline (0.93×–0.99×); the confidence intervals in Figure 1 show that the sub-unity network cases are small but significant, whereas `caso_centurion4` lies within run-to-run noise. The single outlier is `teste1_2zonas-2VGLS-2` on M4 (0.12×): inspecting the runtimes shows this stems from an anomalously short M4 baseline run rather than a regression in the modified version — the optimized M4 runtime (≈112 s) is consistent with the x86-64 optimized runtime; we flag it for further investigation.

5. Discussion

On x86-64, GCC maps `long double` to the 80-bit x87 format, which has no SIMD support: profiling of the original version shows extended-precision `libm` calls dominating the runtime (over 65% of CPU cycles in `rede_gasoduto`, with `powl` alone near 37–45%). Demoting to `double` lowers the per-call cost and, crucially, unblocks vectorization the compiler cannot apply to x87 — under `-O3 -march=native` we observed the AVX2 vectorized `pow` (`_ZGVdN4vv_pow_avx2`) in the optimized profile — while halving the in-memory footprint (16→8 bytes) and reducing cache misses by roughly

59%. This explains why the largest gains concentrate in the math-bound, long-running cases.

The Apple M4 results provide an important counterpoint. Unlike x86-64 Linux, where GCC maps `long double` to 80-bit extended precision, the macOS ARM64 application binary interface (ABI) defines it as a standard 64-bit IEEE 754 float. Consequently, the transformation on the M4 essentially replaced 64-bit operations with identical 64-bit operations, yielding runtimes close to the baseline. Its impact thus relies strictly on the compiler and the architectural representation of extended-precision types.

The conclusion is restricted to the evaluated workloads and validation criteria: passing the numerical comparison does not prove that `long double` is unnecessary in all Marlim3 scenarios, only that the extra precision gave no observable benefit for the selected cases.

6. Conclusion and Future Work

This paper evaluated demoting extended-precision floating-point constructs to `double` in Marlim3, applied globally to the C++ and Fortran code.

The impact was strongly architecture-dependent: on x86-64 all evaluated workloads improved, reaching 11.69× speedup, while on Apple M4 most remained close to the baseline. All workloads met the validation criteria of Section 3.4, indicating that the transformation preserved the outputs for the selected cases.

Future work includes extending the evaluation to more Marlim3 workloads and to NVIDIA Grace, investigating the Apple M4 slowdown in `teste1_2zonas-2VGLS-2`, comparing the transformation against automated precision-tuning frameworks (e.g., Precimonious), evaluating energy efficiency, and studying selective mixed precision to preserve x86-64 gains with fewer source-code changes.

References

Chiang, W.-F., Baranowski, M., Briggs, I., Solovyev, A., Gopalakrishnan, G., and Rakamaric, Z. (2017). Rigorous floating-point mixed-precision tuning. In *Proceedings of the 44th ACM SIGPLAN Symposium on Principles of Programming Languages*, POPL '17, pages 300–315. ACM.

Goldberg, D. (1991). What every computer scientist should know about floating-point arithmetic. *ACM Computing Surveys*, 23(1):5–48.

Higham, N. J. and Mary, T. (2022). Mixed precision algorithms in numerical linear algebra. *Acta Numerica*, 31:347–414.

Monniaux, D. (2008). The pitfalls of verifying floating-point computations. *ACM Transactions on Programming Languages and Systems*, 30(3).

Petrobras (2026). Marlim3: Petrobras Multiphase Flow Simulator. `https://github.com/petrobras/marlim3`. Accessed: 2026-04-29.

Rubio-González, C., Nguyen, C., Nguyen, H. D., Demmel, J., Kahan, W., Sen, K., Bailey, D. H., Iancu, C., and Hough, D. (2013). Precimonious: Tuning assistant for floating-point precision. In *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, SC '13. ACM.