Numerical Reproducibility: Feasibility Issues

Philippe Langlois, Rafife Nheili, Christophe Denis

Numerical Reproducibility: Feasibility Issues

Philippe Langlois, Rafife Nheili
Univ. Perpignan Via Domitia,
Digits, Architectures et Logiciels Informatiques, F-66860, Perpignan.
Univ. Montpellier II, Laboratoire d'Informatique,
Robotique et de Microélectronique de Montpellier,
UMR 5506, F-34095, Montpellier. CNRS. France.
first_name.last_name@univ-perp.fr

Christophe Denis
EDF R&D, F-92141 Clamart.
CMLA-ENS Cachan,
F-94235 Cachan. France.
first_name.last_name@edf.fr

***Abstract*—Floating-point arithmetic may introduce failures of the numerical reproducibility between *a priori* similar sequential and parallel executions of HPC simulation. We present how to apply some existing techniques to part of hydrodynamic finite element simulations. We analyze how easy these techniques allow us to recover its numerical reproducibility.**

***Keywords*—Numerical reproducibility, floating-point arithmetic, finite element simulation, open Telemac-Mascaret, Tomawac.**

I. Motivations

Exascale high performance computing will soon allow us to process 10^{18} floating-point operations per second and so to enable the numerical simulation of larger, more complex and more sensitive physical phenomena. This huge amount of computing power proceeds from the massive and heterogeneous parallelism of our machines: current HPC computers consist in more than one million of computing units. Failures of the numerical reproducibility of some HPC simulations have been reported in several sensitive application domains as is energy [1], dynamical weather science [2], dynamical molecular [3] or dynamical fluid [4]. Indeed HPC is performed in a finite precision world where for instance, the floating-point addition is no more an associative operation. So computed values depend on the order of the operations and may be non-reproducible between *a priori* similar executions.

We illustrate a reproducibility failure case with a HPC code for the simulation at the industrial scale of free-surface flows in 1D-2D-3D hydrodynamic. The open Telemac-Mascaret suite is an integrated set of open source Fortran 90 modules [5]. It includes more than 300 000 lines of code issue from a 20 years international collaboration and declares 4000 registered users. The 2D-simulation of the Malpasset dam break (433 dead people and huge damage in 1959) is performed with a finite element resolution of the Saint-Venant equations. Unknowns are the water depth (H) and its velocity (U,V). The parallel resolution uses a subdomain decomposition of the triangular element mesh (26000 elements, 53000 nodes, 2200 seconds with a 2 sec. time step). Table I exhibits the non-reproducibility of some randomly chosen point values with respect to the number of computing units: 1 vs. 64 vs. 128. Most of the parallel results share nothing more than the order of magnitude compared to the sequential result.

Table I. Reproducibility failure of the dam break simulation

	The sequential run	a 64 procs run	a 128 procs run
depth H	0.3500122E-01	0.2748817E-01	0.1327634E-01
velocity U	0.4029747E-02	0.4935279E-02	0.4512116E-02
velocity V	0.7570773E-02	0.3422730E-02	0.7545233E-02

Numerical reproducibility is necessary to debug or to validate the computed simulation. Reproducibility may also be mandatory to satisfy legal agreements. The comparison between trustful sequential results and parallel ones is a common practice to check the correctness of the parallel implementation. How to conclude for our previous example? Of course, non reproducible executions may be the symptom of remaining bugs. But how to fix them if errors cannot be reproduced? In most cases, such failures damage the confidence in the whole numerical simulation process.

The operation order uncertainty for consecutive executions of *a given binary* file explains this non reproducible behavior. It appears both in parallel or in sequential+vectorized (SIMD) environments. One source is the parallel loop reduction provided by SIMD, openMP, MPI or GPU programming. The number of computing units p modifies the partial computed values before the reduction. Even for a given p, the computed reduced value depends on the dynamic scheduling of the reduction (openMP, MPI, GPU). One other source is the memory data alignment. This appears both in the parallel and the sequential case. Different p modifies the data alignment to the cache line boundary. Hence the vectorized iteration loop and the prologue or the epilogue parts apply to different numerical values. Sequential cases where this alignment depends on

external events coming from the operating system have even been reported [6] .

The reproducibility requirement is not a portability issue. Portability problems comes when one source code yields different binaries, *i.e.* binaries with different numerical properties. In our scope, the main portability parameters are the compilers, their options, the libraries, the few freedom spaces let by the IEEE-754 floating-point standard (rounding modes, double rounding, fused multiply and add operator) and the targeted computing unit. As mentioned before, reproducibility may fail for a given set of these portability parameters.

As already pointed out by [7], the reproducibility requirement is not a claim for accuracy. Numerical reproducibility is getting bitwise *identical* results for every p-parallel run, $p \geq 1$. Full accuracy, or accuracy up to the computing precision, is getting bitwise *exact* result. [7] introduces reproducible but not necessarily accurate summation algorithms. Of course, improving the computed result accuracy up to the IEEE-754 correct rounding guarantees its numerical reproducibility.

In this paper, we study the feasibility issues of numerical reproducibility. Do existing techniques *easily* provide reproducibility to large software used for industrial scale simulations? In Section II we present one test case from the previously introduced hydrodynamic simulation code Telemac. In Section III we apply three existing scenarios to recover the numerical reproducibility of a simulation. A first conclusion is that all presented solutions succeed. This motivates us to describe how easy, or not, these reproducible simulations have been derived from the original code. Some recent algorithms specifically designed to ensure reproducible summation surprisingly appears to be, at least in our context, both the less easily applicable solution and the most expensive one. The chosen test case is a real simulation at the industrial scale. So the presented results are significant and encouraging. Nevertheless it only covers a small part of a general open Telemac-Mascaret treatment and a large amount of work remains to be produced. This also justifies this preliminary study that will help to identify the best choices for the future steps towards trustful and reproducible large scale simulations.

II. One test case and its analysis

A. The test case and the original floating-point computation

We consider Tomawac, the Telemac's module for the wave propagation simulation in coastal areas. It solves a transport equation, a first order PDE changed as ODEs along its characteristic curves [8]. This turns to solve a diagonal linear system where the second member comes from a finite element assembly. Unknowns here are the significant wave height, the mean wave frequency and direction. We solve the Tomawac's Nice test case which simulates the effect of high-speed ferry waves when approaching the Mediterranean Nice harbour. The relative variations between the sequential execution (FPAss) and the p-parallel (FPAss^p) ones are measured at every mesh node for every time step for $p = 2, 4, 8, 16$. The non reproducibility of these computations is illustrated with Figure 1 which displays $\max |\text{FPAss}^p\text{-FPAssl}/|\text{FPAssl}$. The dashed line plots the computing precision (IEEE-754 binary64).

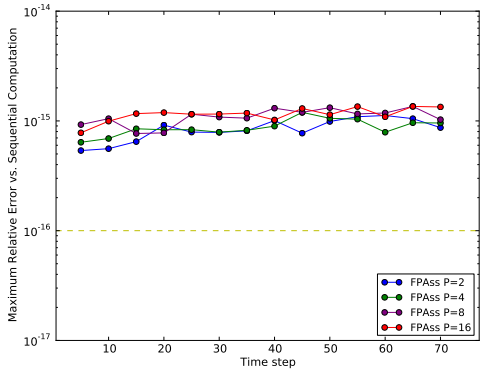


Figure 1. Floating-point assembly. Mean frequency wave, Tomawac (Nice)

B. Analysis and solutions

The main computing step is the finite element assembly of the elementary contributions W_e to every mesh node. For every mesh element e that contains the i node, it consists in the accumulation:

$$X(i) = \sum_{\text{elements } e} W_e(i). \tag{1}$$

The parallel resolution divides the mesh into subdomains that are distributed over the computing units. Subdomain decomposition introduces inner and interface nodes, the latter belonging to a common boundary between several subdomains, *i.e.* between several computing units. Inner node values $X(i)$ are computed as previously while the interface nodes needs a two step accumulation: Relation (1) applied over the elements of one subdomain d_k gives $X^{d_k}_{\text{sub}}(i)$. Then communications between the subdomains that contain the interface node i allows us to compute the final contribution:

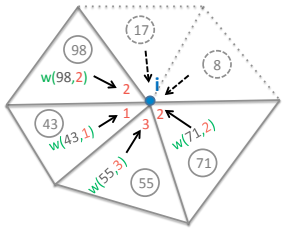
$$X(i) = \sum_{\text{subdomains } d_k} X^{d_k}_{\text{sub}}(i). \tag{2}$$

The parallel assembly step adds the communication and the reduction (2) of the subdomain's $X^{d_k}_{\text{sub}}(i)$ for every interface node i. In practice, every subdomain uses a local table that defines its communication scheme. For instance subdomain d_k knows how much, which i and to which $d_{k'}$, it has to send $X^{d_k}_{\text{sub}}(i)$ and to receive $X^{d_{k'}}_{\text{sub}}(i)$. The Tomawac's implementation of this communication scheme introduces a different accumulation order of a given i according to the subdomains. For $p = 3$ subdomains, the d_0's computation of $X(i)$ is:

$$X(i) = X^{d_0}_{\text{sub}}(i) + X^{d_1}_{\text{sub}}(i) + X^{d_2}_{\text{sub}}(i),$$

while the d_1 and d_2 ones are:

$$X(i) = X^{d_1}_{\text{sub}}(i) + X^{d_0}_{\text{sub}}(i) + X^{d_2}_{\text{sub}}(i),$$




```
for idp = 1, ndp
  for ielem = 1, nelem
    i = IKLE(ielem,idp)
    X(i) = X(i) + W(ielem, idp)
```

Figure 2. Assembly step at the inner node i and its associated numberings.

and

$$X(i) = X^{d_2}_{\text{sub}}(i) + X^{d_0}_{\text{sub}}(i) + X^{d_1}_{\text{sub}}(i).$$

Hence the floating-point computed $X(i)$ may differ over the subdomains. Let us remark that the dynamic scheduling of classical reductions may also generate the same differences. To ensure the solution continuity between the subdomains, Tomawac introduces one more communication step to share the maximum value of every $X(i)$ (this choice being justified by physical reasons). As a side effect, this ensures the reproducibility between repeated simulations at a given p (which is not the case with the dynamic reduction). Nevertheless this is not enough to recover the simulation reproducibility for different p: in the previous relations, $X(i)$ depends on the value p.

Before introducing three possible scenarios to ensure the numerical reproducibility, we present a practical but important assembly step feature. The well known implementation difficulty of these steps comes from the management of the node numbering. Finite element assembly introduces several numberings for a given node and tables to map from one to another. Figure 2 describes it: every triangular element ielem has ndp=3 nodes. Assembly loops over the domain or the subdomain elements, and then over the element nodes. The IKLE table returns the global number i of the node related to the contribution $W_e(i)$ in (1). Let us note it defines indirect accumulations, *i.e.* two consecutive iterations of the inner loop do not apply to the same i value.

III. Evaluation of three reproducible solutions

We propose to evaluate the following three existing techniques that provide reproducibility of this assembly step. Since here this latter is not ill-conditioned, the compensated summation of [9] yields a correctly rounded accumulation, and so an accurate and reproducible computation. Another choice are recent Demmel and Nguyen's reproducible sums [7]. The latest Telemac release introduces integer conversions and so exact accumulations [5].

We use the following notations to measure the accuracy and the reproducibility of these solutions. A^s denotes a sequential assembly algorithm and A^p its p-parallel version. With $\max_{rel}(A_1, A_2) = |A_1 - A_2|/|A_2|$, the measures of the A^p accuracy compared to the original sequential FPAss^s and of the reproducibility between parallel executions are respectively:

$$\text{acc} = \max_{rel}(A^p, \text{FPAss}^s),$$

$$\text{rep} = \max_{rel}(A^p, A^s).$$

A. Accurate compensated assembly

Compensated summation consists in computing every rounding error generated by the successive floating-point additions and to accumulate them in an error term. The addition rounding error is actually a floating-point number that can be exactly computed with the Knuth-Moller's 2Sum algorithm or (in base 2) with the Dekker-Kahan's Fast2Sum one, *e.g.* see [10]. The rounding-error accumulation is performed in parallel along the main accumulation. Hence the computation of the error term does not need the knowledge of the whole sum entries to start. This error term accumulation performed in floating-point arithmetic again generates rounding-errors but the same principle iteratively applies if necessary. Finally this error term is added to the main sum value and so compensates it. Ogita-Rump-Oishi prove that one compensation iteration improves the sum's accuracy as if this sum was computed in twice the computing precision $\mathbf{u}$ [9]. For condition numbers less than $1/\mathbf{u}$ and reasonable sum lengths, the compensated algorithm returns a correctly rounded sum. It is our case here.

Compensated assembly steps are easy to derive for both inner and interface node assembly steps. Accumulation (1) is now written as:

$$[X(i), E(i)] = \text{Compensated Sum}_{\text{elements } e} W_e(i),$$

where $E(i)$ is the accumulated error term associated to $\sum_{\text{elements } e} W_e(i)$. The compensated assembly of an inner node is:

$$X(i) + E(i),$$

while the interface node communications encompass the pairs $[X^{d_k}_{\text{sub}}(i), E^{d_k}(i)]$. The compensated assembly of an interface node now is:

$$X(i) + G(i),$$

where the error term $G(i)$ carefully takes into account the two step accumulation as follows. *i)* $F^{k,k'}(i)$ is the rounding error generated by the partial accumulation in (2) over two consecutive subdomains d_k and $d_{k'}$ that share the interface node i: $X^{d_k}_{\text{sub}}(i) + X^{d_{k'}}_{\text{sub}}(i)$. *ii)* The whole error term in (2) is then computed, according to the parenthesis order, as:

$$G(i) = \sum_{\text{subdomains } d_k, d_{k'}} (E^{d_k}(i) + E^{d_{k'}}(i)) + F^{k,k'}(i).$$

This compensated assembly fits well the existing one. It follows the iteration flow defined in Figure 2 for the inner nodes (Relation (1)) and for the interface ones (Relation (2)). We already noted the very slight modification of the communications between the computing units that now exchange the

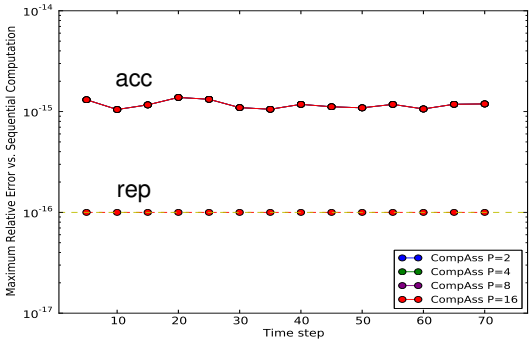


Figure 3. Compensated assembly. Mean frequency wave, Tomawac (Nice)

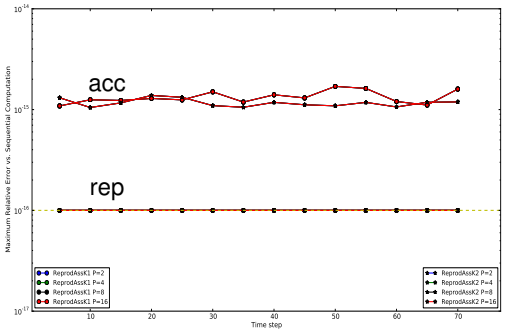


Figure 4. Reproducible based assembly. Mean frequency wave, Tomawac (Nice)

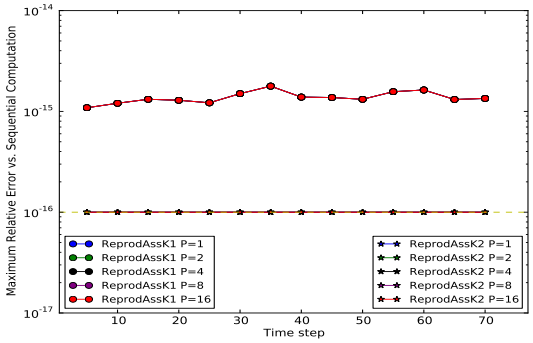
pair of floating point value $[X_{\text{sub}}^{d_k}(i), E^{d_k}(i)]$, and not only its first one. Nevertheless no supplementary communication step is introduced here.

Tomawac recovers numerical reproducibility with this compensated assembly. Figure 3 exhibits that all parallel executions return the same simulation results up to the computing precision (rep for $p = 2, 4, 8, 16$). Since compensation also improves the accuracy, the acc plot actually measures the original accuracy of the floating point assembly FPAss^s. It measures about one digit enhancement which is consistent with the error bound of the classical floating point accumulation. Compensation introduces a small amount of supplementary computation but we did not measure any significant increase of the whole simulation time.

B. Reproducible algorithm based assembly

Demmel and Nguyen have recently introduced algorithms to compute reproducible sums, *i.e.* bitwise identical results independently on the summation order [7]. These solutions derive from Rump-Ogita-Oishi's AccSum and FastAccSum algorithms [11]. A pre-rounding step returns K shrunks over the entries such that the floating-point sum of each shrunk is exact. Defining the right shrunk cutting width is the key-point of this error-free vector transformation of the entries: it depends on the maximum absolute value of the entries and of their numbers. These ReproSum and FastReprodSum algorithms are parallel K-fold processes where K is *a priori* chosen; the larger K, the more accurate sum. In practice, they introduce two reductions ($\max$ and sum).

The main difficulty to integrate these algorithms within the assembly step in Tomawac is the need for maxima of the inner point and the interface point contributions, respectively $\max_e W_e(i)$ in (1), and one other maximum value for (2) we detailled a few lines further. As we already mentioned it, the assembly loop (Figure 2) consists in indirect accumulations. In this scope, ReprodSum and FastReprodSum introduces a significant volume of supplementary computations and communications. One more previous run of the assembly double loop is necessary to identify the maximum values for every inner node i (and also generates some extra-storage



;

Figure 5. Accuracy of the reproducible based assembly for $K = 1, 2$ compared to the compensated one. Mean frequency wave, Tomawac (Nice)

on every computing unit). The interface node assembly is even more costly since every computing unit needs to use the same maximum value before computing both Relation (1) and Relation (2). For every subdomain d_k, $\max_{e_k \in d_k} W_{e_k}(i)$ is first identified and reduced over the domain before computing every $X_{\text{sub}}^{d_k}(i)$ with Relation (1). Relation (2) is finally computed after a last classical communication step of the $X_{\text{sub}}^{d_k}(i)$. Hence the interface node assembly adds one other previous run similar to the Relation (2) implementation and the reduction of the identified partial maximum values.

Nevertheless the reproducibility failure of the floating point assembly is thus repaired for all considered parallel executions. Figure 4 exhibits it for $K = 1$ and $K = 2$. Figure 5 presents the accuracy behavior of these solutions for $K = 1$ and 2 compared to the accurate compensated assembly. As expected, the $K = 1$ case provides less accurate results, without being here significantly interesting even compared to the original floating point ones. It also illustrates that no difference appears for $K = 2$ between the ReprodSum based assembly and the accurate compensated one.

C. Integer converted assembly

The last solution towards reproducible assembly step is the integer conversion the latest Telemac release provides [5]. It

consists in one 8 byte integer conversion of every floating point value concerned by assembly computations. Integer accumulations are exact as long as no overflow occurs and their conversions back to floating point values yield reproducible computed results.

This 8 byte integer conversion is simple. We denote $\max(INTK8)$ the maximum of these 8 byte integer values. A N-length floating point vector X is converted into the integer vector:

$$IX = INT(X \times Q_8(X, N)), \tag{3}$$

where the scaling factor Q_8 is chosen to avoid the overflow while summing X. We have [5]:

$$Q_8(X, N) = \frac{\max(INTK8)}{N \times \max(|X|)}.$$

This conversion introduces two approximation levels. The first one comes from the difference between the floating-point discretisation and the integer one. We illustrate it with one example in decimal. Let $N = 10$ and $X \in [10^2, 10^6[$ (componentwisely); so $\max(X) = 10^6$. Let $IX \in [0, 10^8[$ (componentwisely) thus $\max(IX) = 10^8$. The conversion $IX = X \times Q$ applies here with $Q = 10$. So every floating point value with more than one fractional digit suffers from a 10^{-1} absolute error of conversion. This corresponds to a relative error from 10^{-3} to 10^{-7} for the whole X range that acts as data errors before the summation. Even if the sum is exactly computed, cancellation is well known to magnify the data errors. Hence no accuracy bound better than 10^{-4} can be expected after such (fancy) conversion. The floating point evaluation of the scaling factor Q_8 introduces a second approximation level. Indeed evaluation (3) suffers from rounding errors in the Q_8 evaluation, in the multiplication and then in the division on the way back to floating point values. Of course, these errors are smaller than the previous discretisation ones but occur in every conversion (3). Nevertheless changing Q_8 to its next larger power of the base, *i.e.* to 2^α with $2^{\alpha-1} < Q_8 \leq 2^\alpha$, in binary arithmetic, is a slight modification that avoids this approximation error. In this case, it is clear that no difference appears between the exact floating-point sum and its integer counterpart as long as $X \times 2^\alpha \in INT\ K8$.

There is no difficulty to apply this integer conversion to the assembly process in Tomawac. The pre and post conversions are applied on-the-fly while accumulating inner nodes in (1). Interface assembly step only needs to communicate the integer values to evaluate (2).

Integer conversion yields the expected reproducibility of the simulation as exhibited with Figure 6. It does not appear a significant loss of accuracy compared to the previous solutions. The Tomawac computation is a favourable case for this integer conversion strategy. Indeed few terms are accumulated for every node i and their ranges remain small enough to avoid suffering a lot from the discretisation error.

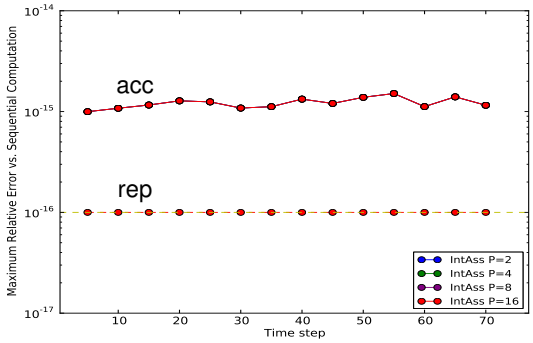


Figure 6. Integer assembly. Mean frequency wave, Tomawac (Nice)

IV. Conclusion and future work

We have been able to recover the numerical reproducibility of the Tomawac simulation with the three presented solutions. In this context we can summarize their current feasibility as follows. The compensated solution appears to be the easiest one to apply and provides accurate results for a low computing over-cost. The integer conversion provided in Tomawac is also easy to derive and introduces a low over-cost. Even if it is definitely not redhibitory here, this solution may introduce a risky loss of accuracy in other cases. A careful error analysis should be performed before other applications. The solution that uses the reproducible sums is efficient but applies less easily to our case and introduces a significant communication over-cost. Let us mentioned that the latest evolution of these algorithms only introduces one reduction step [12] and so should introduce less communication cost in our context.

A preliminary step has been to identify what parts of such large computation generate failures of the numerical reproducibility while changing the number of computing resources. Assembly is an inevitable step in the ubiquitous finite element resolution. The chosen simulation was simple enough to successfully only focus on this assembly step. Future work will tackle the linear system resolution which is the second main step of such simulations. The efficiency, in term of computing time overcost, will be of most interest for this next stage.

V. Acknowledgment

Authors thank J.-M. Hervouet, EDR R&D (Chatou, France) for his valuable comments and his strong support to this work.

References

[1] O. Villa, D. G. Chavarría-Miranda, V. Gurumoorthi, A. Márquez, and S. Krishnamoorthy, "Effects of floating-point non-associativity on numerical computations on massively multithreaded systems," in *CUG 2009 Proceedings*, 2009, pp. 1–11.

[2] Y. He and C. Ding, "Using accurate arithmetics to improve numerical reproducibility and stability in parallel applications," *J. Supercomput.*, vol. 18, pp. 259–277, 2001. [Online]. Available: http://dx.doi.org/10.1023/A%3A1008153532043

[3] M. Taufer, O. Padron, P. Saponaro, and S. Patel, "Improving numerical reproducibility and stability in large-scale numerical simulations on gpus," in *IPDPS*. IEEE, 2010, pp. 1–9.

[4] R. W. Robey, J. M. Robey, and R. Aulwes, "In search of numerical consistency in parallel programming," *Parallel Comput.*, vol. 37, no. 4-5, pp. 217–229, 2011.

[5] "Open TELEMAC-MASCARET. v.7.0, Release notes," www.opentelemac.org, 2014.

[6] J. M. Corden and D. Kreitzer, *Consistency of Floating-Point Results using the Intel Compiler or Why doesn't my application always give the same answer?*, Intel Corporation, Aug. 2014.

[7] J. W. Demmel and H. D. Nguyen, "Fast reproducible floating-point summation," in *Proc. 21th IEEE Symposium on Computer Arithmetic*. Austin, Texas, USA, 2013.

[8] J.-M. Hervouet, *Hydrodynamics of free surface flows: Modelling with the finite element method.* John Wiley & Sons, 2007.

[9] T. Ogita, S. M. Rump, and S. Oishi, "Accurate sum and dot product," *SIAM J. Sci. Comput.*, vol. 26, no. 6, pp. 1955–1988, 2005.

[10] J.-M. Muller, N. Brisebarre, F. de Dinechin, C.-P. Jeannerod, V. Lefèvre, G. Melquiond, N. Revol, D. Stehlé, and S. Torres, *Handbook of Floating-Point Arithmetic*. Birkhäuser Boston, 2010.

[11] S. M. Rump, "Ultimately fast accurate summation," *SIAM J. Sci. Comput.*, vol. 31, no. 5, pp. 3466–3502, 2009.

[12] J. W. Demmel and H. D. Nguyen, "Fast reproducible floating-point summation," Department of Computer Science, University of California, Tech. Rep., 2015.