Recovering numerical reproducibility in hydrodynamic simulations

Philippe Langlois, Rafife Nheili, Christophe Denis

Recovering numerical reproducibility in hydrodynamic simulations

Langlois Philippe,Nheili Rafife
Univ Perpignan Via Domitia
Digits, Architectures et Logiciels Informatiques, F-66860,Perpignan
Univ. Montpellier 2, Laboratoire d'informatique,
Robotique et de Microélectronique de Montpellier,
UMR 5506, F-34095, Montpellier. CNRS. France.
Email: first_name.last_name@univ-perp.fr

Christophe Denis
ENS Cachan,
CMLA Research Center for Applied Maths,
F-94235 Cachan. France.
Email: Christophe.Denis@cmla.ens-cachan.fr

***Abstract*—HPC simulations suffer from failures of numerical reproducibility because of floating-point arithmetic peculiarities. Different computing distributions of a parallel computation may yield different numerical results. We are interested in a finite element computation of hydrodynamic simulations within the openTelemac software where parallelism is provided by domain decomposition. One main task in a finite element simulation consists in building one large linear system and to solve it. Here the building step relies on element-by-element storage mode and the solving step applies the conjugated gradient algorithm. The subdomain parallelism is merged within these steps. We study why reproducibility fails in this process and which operations have to be corrected. We detail how to use compensation techniques to compute a numerically reproducible resolution. We illustrate this approach presenting the reproducible version of hydrodynamic simulations for one test cases provided with the openTelemac software suite.**

***Keywords*—** Numerical reproducibility, finite element, domain decomposition, hydrodynamics simulation, HPC, compensated algorithms, openTelemac.

1. Introduction

High performance computing leads to new scientific discoveries through complex and sensitive numerical simulations. The ability to reproduce these simulation results becomes a crucial property to improve the confidence in large scale numerical experiments. Reproducibility is also required to facilitate the debug, the validation and the test of these huge codes. The changes in a simulation outputs must be directly linked to the changes in some of the simulation parameters, and not accidentally affected by uncontrolled floating-point calculations. Indeed the rounding errors inherent to the floating-point arithmetic and the dynamic reductions of parallel executions modifies the numerical outputs.

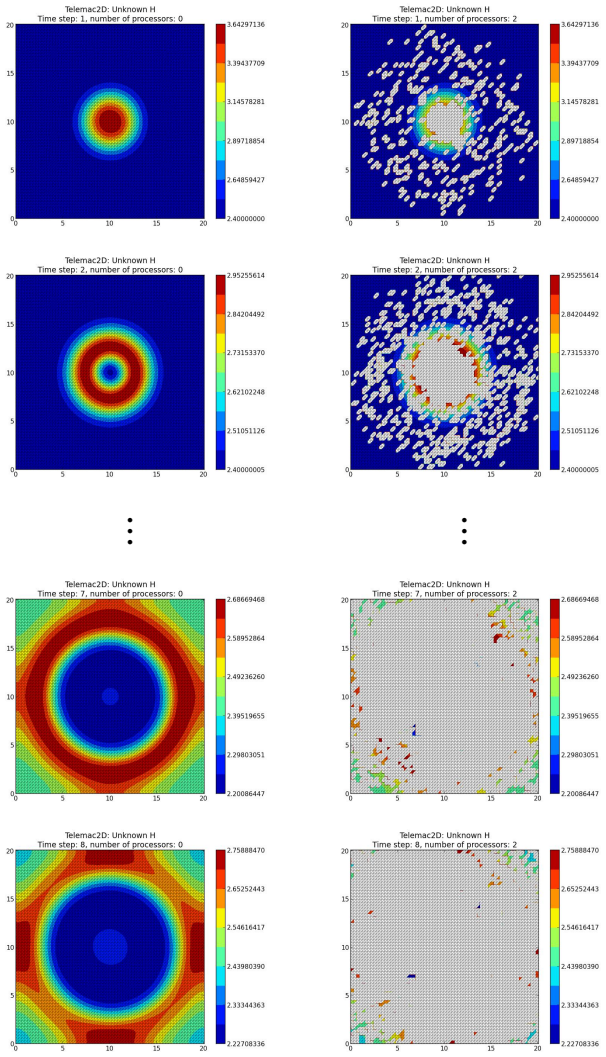
Numerical reproducibility is getting bitwise *identical* for every p-parallel run where $p > 1$ counts the number of computing units. This requirement is not a claim for accuracy. Results could be reproducible but not accurate. Full accuracy is getting bitwise *exact* result, which corresponds to the *correctly rounded* one in the IEEE-754 scope. Of course, correct rounding ensures numerical reproducibility.

A sequential run of a large scale simulation code introduces many rounding errors and suffer from other approximations. Nevertheless this approximate sequential result is prevailing in practice: when the sequential and the parallel simulations disagree, the latter is considered to be the culprit and the former as the reference result. So a convincing modified reproducible code should satisfied two criteria: *i)* bitwise *identical* result for every p-parallel run where $p \geq 1$; *ii)* reproducible result within a reasonable range of differences compared to the original sequential simulation ones. Hence the original sequential simulation and the sequential run of the reproducible one ($p = 1$) could exhibit a reasonable relative difference.

We start exhibiting one typical reproducibility failure case. It concerns the simulation at the industrial scale of free surface flows in 1D-2D-3D hydrodynamic. This simulation is processed with the open Telemac-Mascaret suite which is an integrated set of open source Fortran 90 modules. It consists in more than 300,000 lines of code issue from a 20 years of international collaboration and it declares 4000 registered users [10]. The gouttedo test case, available in the distribution, is the 2D-simulation of a water drop fall in a square basin. This resolution uses a triangular element mesh (8978 elements, 4624 nodes) and simulates several time steps of 0.2 sec.

Figure 1 exhibits the non reproducible behavior of the water depth simulation between the sequential and 2 processor runs. The left plot shows the water depth values returned by the sequential simulation. The right plot is related to the parallel run ($p = 2$). White plots exhibit the mesh elements that differ from the sequential ones. The loss of reproducible values increases as the simulation runs in the time scale since time step t values depends on the previous one.

Figure 1: gouttedo: white plots are non reproducible water depth values between the sequential (left) and a 2 processors run (right). Time steps: 1, 2, ..., 7, 8.



Failures of numerical reproducibility have been reported in various application domains of HPC simulation like in energy [13], dynamical weather science [4], dynamical molecular [12] or dynamical fluid [11] for instance. The first task towards reproducibility is to carefully identify its failure sources and to apply as few as possible corrections to manage the inherent over cost. Two types of solutions exist in the literature. Some recover reproducibility without improving the accuracy. For example, the effects of non-associative floating-point arithmetic is avoided implementing deterministic parallel reduction trees or conversions to fixed-point numbers or integers. Some others rely on accuracy improvement as for example, using compensation algorithms that accumulate and correct all the relevant rounding errors. The results we present hereafter belong to this scheme. Solutions with double-double or quad-double libraries have also been successfully applied. A last and original way to reproducibility merges the two previous ones. In [3], Demmel and Nguyen propose algorithms that provide reproducible sums and allow the user to increase the accuracy depending on the application requirements (by reducing the efficiency and vice versa).

In this paper, we present how to benefit from compensation techniques to recover the numerical reproducibility in the open Telemac-Mascaret suite. The difficulty is not the compensation by itself since this technique is well known by the computer arithmetic community, nor how to introduce it, but where to introduce it. Indeed this question becomes an actual challenge since the finite element simulation and its implementation in a software that targets real life or large scale applications is a complex case study.

The sequel is organized as follows. Section 2 presents a brief overview of the main computation steps in this kind of simulation: the building and the solving phases of the finite element linear system. In Section 3, we exhibit the non reproducible computing sequences and define the compensated alternatives that will gradually lead to parallel reproducible simulations. We successively present how to compensate the interface node assembly introduced by the parallel domain decomposition, the classical finite element assembly and some algebraic transformations that simplify the linear system. The last step is the resolution that mainly consists here in applying the conjugate gradient algorithm. We conclude presenting the reproducible simulation of the gouttedo test case and some first comments.

2. Overview of the finite element computation

In this paper, we consider Telemac2D which is the open Telemac-Mascaret 2D hydrodynamics module. Its main application is in free-surface maritime or river hydraulics [5]. It solves the Saint-Venant equations using the finite element method and a triangular element mesh. At every mesh point, the simulation calculates 3 unknowns: the water depth H and the 2D velocity components U, V. Finite element method leads to build and solve a general sparse linear system:

$$\begin{pmatrix} A_{hh} & A_{hu} & A_{hv} \\ A_{uh} & A_{uu} & 0 \\ A_{vh} & 0 & A_{vv} \end{pmatrix} \begin{pmatrix} H \\ U \\ V \end{pmatrix} = \begin{pmatrix} B_h \\ B_u \\ B_v \end{pmatrix}. \qquad (1)$$

Zero sub-matrices correspond to the coupling absence between the two velocity components.

In practice, one finite element simulation depends on many physical and numerical parameters. For instance, [5] uses the wave equation to decouple water depth and velocity in (1) and introduces diagonal matrices $A2$ and $A3$. The actual system to solve is simplified as $AX = B$:

$$\begin{pmatrix} A1 & 0 & 0 \\ 0 & A2 & 0 \\ 0 & 0 & A3 \end{pmatrix} \begin{pmatrix} H \\ U \\ V \end{pmatrix} = \begin{pmatrix} C1 \\ C2 \\ C3 \end{pmatrix}. \qquad (2)$$

Hence the actual building of System (2) includes the following algebraic transformations of the matrix and the second member:

$$\begin{aligned} A1 &= A_{hh} - A_{hu}A2^{-1}A_{uh} - A_{hv}A3^{-1}A_{vh}, \\ C1 &= B_h - A_{hu}A2^{-1}B_u - A_{hv}A3^{-1}B_v, \\ C2 &= B_u - A_{uh}H, \\ C3 &= B_v - A_{vh}H. \end{aligned} \tag{3}$$

The parameters of the presented test case (gouttedo) are the following. The A matrix is stored element-by-element (see Section 3.2). System (2) is solved in two steps: H is first computed applying the conjugate gradient method to $A1H = C1$; then U, V derive from H thanks to diagonal $A2$ and $A3$.

Parallelism in open Telemac-Mascaret relies on domain decomposition method that splits the building and the resolution phases to distribute them to the available computing units. The number of subdomains modifies the computation. This is a first source of reproducibility failure since the generated rounding-errors differ as the subdomain number varies. On another hand, parallel libraries (as MPI here) also introduce indeterminate order parallel reductions. This also generates non-reproducible algebraic operations since floating-point addition is not associative. These two different sources of non reproducibility affect the parallel implementation of the building and the solving phases of system (2). Hence we aim, first to obtain reproducible matrix A and second member B, then to reproducibly compute its solution X.

Previous results exhibit that compensated summation algorithms yield an efficient and numerically reproducible finite element assembly step applied to one single vector [8]. These well known algorithms, based on the early works of Knuth [7], Kahan [6], and Dekker [2], improves the sum accuracy as if it was computed in twice the working precision $\mathbf{u}$ [9]. So compensated summation returns a correctly rounded sum for reasonable sum lengths with a condition number smaller than $1/\mathbf{u}$. It consists to compute every rounding error generated by the successive floating-point additions and to accumulate them in one error term that is finally added to compensate the sum. In the sequel, we apply these compensation techniques to recover the reproducible building and resolution of System (2).

3. Steps towards reproducible finite element resolution

In this section, we detail the computation steps corrected to recover a reproducible finite element resolution. The core of this resolution is the assembly process which strongly differs between sequential and parallel implementations. The latter one introduces interface nodes that are assembled during the building and the solving phases of system (2). We detail each of these steps to enlighten their non-reproducibility sources and then we define how to correct it.

Table 1 aims to help the reader to follow the reproducibility enhancement of system (2) during the building and the solving phases. Its first column exhibits that all the components of system (2) suffer from non reproducibility in the original Telemac2d computation.

	original	after the building phase	after the H solving	after the U,V solving
A1	✗	✗	✗	✗
A2	✗	✓	✓	✓
A3	✗	✓	✓	✓
C1	✗	✓	✓	✓
C2	✗	✓	✓	✓
C3	✗	✓	✓	✓
H	✗	✗	✓	✓
U	✗	✗	✗	✓
V	✗	✗	✗	✓

Table 1: Reproducibility enhancement steps of system (2) components in gouttedo

The parallel resolution divides the mesh into subdomains that are distributed over the computing units. Domain decomposition introduces inner and interface nodes, the latter belonging to a common boundary between several subdomains, *i.e.* shared between several computing units. In the whole section, V denotes an arbitrary vector extracted from system (2) and defined for every domain node both (inner or interface ones).

Our notations means that relations and operations with V apply to the whole vector V while those with $V(i)$ aim to identify separate processes with respect to the node type (inner or interface ones).

3.1. Interface node assembly

The interface node assembly is one the main significant differences between the sequential and the parallel resolutions. We start to focus on this interface node assembly that greatly affects the reproducibility.

3.1.1. Original computation. Let V be an arbitrary vector extracted from system (2) and defined for every domain node. Let i be one interface node that belongs to several subdomains d_k. Let $V^{d_k}(i)$ be the contribution of the d_k subdomain to V at the interface node i (the computation of V^{d_k} only includes quantities related to d_k). Communications between the subdomains d_k yield the global contribution $V(i)$ at the node i as the following reduction:

$$V(i) = \sum_{\text{subdomains } d_k} V^{d_k}(i). \tag{4}$$

This reduction occurs for every interface node i.

In practice here, 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 $V^{d_k}(i)$ and to receive $V^{d_{k'}}(i)$. For a given i, the Telemac2D implementation of this communication scheme introduces different, but statically defined, accumulation order with respect to the subdomains. For instance when $p = 3$

subdomains, the d_0's computation of $V(i)$ with Relation (4), denoted as $V(i)|_{d_0}$, is:

$$V(i)|_{d_0} = V^{d_0}(i) + V^{d_1}(i) + V^{d_2}(i),$$

while the d_1 and d_2 ones are:

$$V(i)|_{d_1} = V^{d_1}(i) + V^{d_0}(i) + V^{d_2}(i),$$

and

$$V(i)|_{d_2} = V^{d_2}(i) + V^{d_0}(i) + V^{d_1}(i).$$

Each reduction is statically ordered according to the increased numbering of the neighbouring subdomains. Hence the floating-point computed $V(i)|_{d_k}$ may differ over the subdomains d_k. Nevertheless the static strategy ensures the reproducibility between repeated simulations for a given number of computing units p, that is not the case with the classical dynamic reduction.

To recover the solution continuity between the subdomains, *i.e.* $V(i)|_{d_k} = V(i)|_{d'_k}$, Telemac2D introduces one more communication step to share the maximum value of every $V^{d_k}(i)$ (this choice is justified by physical reasons). Nevertheless this is not sufficient to recover reproducibility for runs with different numbers of computing units: the exchanged $\max_{d_k} V(i)|_{d_k}$ depends on the number of d_k that splits accumulation (4).

3.1.2. Reproducible computation. These computed $V(i)$ differ as the subdomain number varies since accumulations (4) generate different rounding errors. We recover reproducibility using compensated algorithms: these different sets of rounding errors are taken into account such that the remaining rounding error in the compensated $V(i)$ does not depend anymore of the number of subdomains. Here compensation consists in accumulating every generated rounding error until Relation (4) achieves to compute the interface node assembly. Hence every member V now comes from and goes with its accumulated rounding error E_V until the last reduction in (4), after which E_V compensates V. Compensating the accumulation (4) of the interface node assembly now applied to the pairs (V, E_V) is easy. For all subdomains $d_k \in D_i$ that share the interface node i, we write:

$$V(i), E_V(i) = \bigoplus_{\text{subdomains } d_k} V^{d_k}(i), E_V^{d_k}(i). \qquad (5)$$

This compensated sum uses the classical 2Sum error-free transformation similarly as in Sum2 [9]: for every $k \in D_i$, we compute:

$$\begin{aligned} V(i), e_k &= \mathsf{2Sum}_k(V(i), V^{d_k}(i)), \\ E_V(i) &= E_V(i) + E_V^{d_k}(i) + e_k. \end{aligned}$$

Step (6) accumulates V and computes each corresponding generated rounding error e_k. Step (6) accumulates e_k and the previous errors $E_V^{d_k}(i)$ in $E_V(i)$.

Finally, the compensated vector is computed after the last reduction of every interface node i to the whole vector V as:

$$V + E_V. \qquad (6)$$

We stress that this final compensation applies to the vector of the inner and interface nodes after the end of the interface node assembly. We include it in this step and describe it in this section to simplify the presentation.

Similarity with double-double arithmetic exists here since entries are also floating-point couples (V, E_V) and computations update the error term E_V. Nevertheless no normalisation (with 2Sum or Fast2Sum) applies here to maintain the relative accuracy between V and E_V. No partial compensation neither applies before Relation (6).

In the following, V equals the vector components of System (2): $A2$, $A3$, $C1$, $C2$, $C3$. Computations (4) and (6) appear differently during the simulation: in the building phase for all except $A1$ and in the resolution phase for the matrix-vector products $\boldsymbol{A1d}$ of the conjugate gradient iterations (see Figure 3). Table 2 exhibits how this interface node assembly is merged within the solving and building phases in the resolution workflow. These two phases are described in the next sections.

Unknowns	H	U	V
System building:			
Finite element assembly (7) & Algebraic transformations (3)	$A2, A3$ $C1(A2, A3)$ $A1(A2, A3)$	$C2(H)$	$C3(H)$
Interface node assembly (4)	$A2, A3, C1$	$C2$	$C3$
System solving:			
	$A1, C1 \rightarrow H$ Conjugate gradient	$C2, A2 \rightarrow U$	$C3, A3 \rightarrow V$
Interface node assembly (4)	In each iteration $\boldsymbol{A1d}$		

Table 2: Transformation workflow in System (2) and related component dependencies in brackets.

3.2. Building the system linear

The building phase of System (2) includes two types of non reproducible operations: the finite element assembly step (of the inner nodes of every subdomain) and the algebraic transformations (3).

3.2.1. Original computation. Assembly and algebraic operations depends on the (finite element) matrix storage mode. Here applies the element-by-element storage (EBE). Matrix-vector product of assembled quantities is the main floating-point process of the building and the solving phases. EBE leads to very efficient matrix-vector products in the finite element context: it avoids to assemble the

whole matrix to compute a matrix-vector product and it also reduces the matrix memory print. Hence EBE storage introduces a specific processing of matrix-vector product (merging product and assembly) that will be studied in the next Section 3.3.

The finite element assembly accumulates the elementary contributions W_e for every mesh element e that contains the i node as:

$$V(i) = \sum_{\text{elements } e} W_e(i). \tag{7}$$

In the parallel resolution, this accumulation is distributed over the subdomains d_k and its computation differs between inner nodes and interface nodes. Relation (7) returns $V(i)$ for every inner node i while for one interface node, it only computes the partial contribution $V^{d_k}(i)$ already defined in Relation (4).

In the following, this assembly is applied to build several vectors in System (2): the second member vectors $C1, C2, C3$, the diagonal of matrix $A1$, the diagonal matrices $A2, A3$, and as we will detail it in Section 3.3, the result of the matrix-product $\boldsymbol{A1d}$ computed in the conjugate gradient iterations.

Table 2 resumes how the finite element assembly applies in the whole resolution. For instance, the second member $C1$ of the H system depends on $A2$, the matrix of the U system (see the second relation in (3)). Moreover, the U system depends on the H value. Hence $A2$ has to be assembled while building the H system.

After the assembly step, algebraic transformations (3) are directly applied to the assembled vectors V. It only includes products between diagonal matrices and vectors, scalar and vector, and vector additions.

3.2.2. Reproducible computation. The elementary contributions W_e in accumulation (7) only depends on the element e and so does not depend on the domain decomposition, *i.e.* suffers from no parallel effect. This accumulation (7) has the same order with respect to e for inner nodes in the sequential and parallel cases. Nevertheless, a given inner node i may become one interface node in another domain decomposition, *i.e.* when the number of computing units varies. Hence we must apply a reproducible assembly to every mesh node both in the sequential and in parallel simulations to avoid a non-deterministic error propagation.

Compensating this vector assembly step is efficient here [8]. As described in Section 3.1, we introduce (V, E_V) pairs and associated computations. A reproducible finite element assembly step is similar to a classical compensated sum, *e.g.* a Sum2 [9]. It computes:

$$V(i), E_V(i) = \text{Compensated Sum}_{\text{elements } e} W_e(i), \tag{8}$$

where $E_V(i)$ is the accumulated error at the node i of the contributions from the elements which it belongs to.

Compensated version of the algebraic transformations 3 are easy to derive computing and propagating the error term E_V. For example, the product between a diagonal matrix D and a vector X is now computed as follows. Assuming D stored as the vector Y, we compute the pair:

$$V(i), E_V(i) = X(i), E_X(i) \otimes Y(i), E_Y(i), \tag{9}$$

defined as:

$$\begin{aligned} V(i), e &= \mathsf{2Prod}(X(i), Y(i)), \\ E_V(i) &= X(i) \times E_Y(i) + Y(i) \times E_X(i) + e, \end{aligned}$$

using the classical 2Prod error free transformation. Others operations in (3) derive similarly.

Finally, the correction of the finite element assembly, the interface node assembly and the algebraic transformations (3) provides the reproducibility of matrices and vectors at the end of the building phase as reported in the second column of Table 1. System (2) is now reproducible except $A1$ and the unknowns. Matrix $A1$ remains non reproducible because it is never assembled for the interface nodes. This will be integrated during the next solving phase assembling vectors $\boldsymbol{A1d}$.

3.3. Solving the linear system

3.3.1. Original computation. As already mentioned, the simulation solves the three subsystems (2). Reproducible U and V derived easily from similar algebraic transformations to those presented in Section 3.2.

Table 3: Conjugate gradient algorithm

Entries:	Matrix A, second member B
	Initialization vector g, one stopping criteria
Output:	Vector X
Initialization:	
	$r^0 = AX^0 - B$, $d0 = g^0$
	$\rho^0 = \dfrac{(r^0, g^0)}{(Ad^0, d^0)}$
	$X^1 = X^0 - \rho^0 d^0$
Iterate until stopping criteria:	
	$r^m = r^{m-1} - \rho^{m-1} \boldsymbol{Ad^{m-1}}$
	$d^m = g^m + \dfrac{(r^m, d^m)}{(r^{m-1}, g^{m-1})} d^{m-1}$
	$\rho^m = \dfrac{(r^m, d^m)}{(d^m, \boldsymbol{Ad^m})}$
	$X^{m+1} = X^m - \rho^m d^m$

The H system is solved with the conjugate gradient algorithm without preconditioning – see Table 3. Dot product are denoted in brackets. The important operation is the matrix-vector product denoted in bold as $\boldsymbol{Ad}$.

Thanks to the EBE storage, one matrix M is decomposed as one assembled diagonal matrix MD and one non-

assembled extra-diagonal matrix MX_e. The matrix-vector product $R = M\ V$ is now computed as:

$$R = MD\ V + \sum_{\text{element } e} MX_e\ V = R_1 + R_2. \qquad (10)$$

Vector $R_1 = MD\ V$ is the product of one assembled diagonal matrix and a vector. The product $MX_e\ V$ is assembled with Relation (7) and yields the vector R_2. Finally the R result is assembled at the interface nodes in each iteration with Relation (4).

As already mentioned, the second members $C2$ and $C3$ of the U and V systems only depends on H. Hence the interface node assembly (4) is applied to $C2$ and $C3$ before the last system resolutions (see Table 2).

3.3.2. Reproducible computation. Now we detail how to recover the reproducibility of the three subsystems.

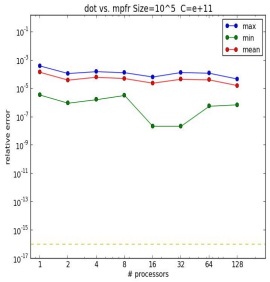
We begin with the H one, $A1H = C1$, which is solved with the conjugate gradient method. The non reproducibility of the output H comes from two sources *i)* the matrix-vector product (bold) and *ii)* the scalar product. In Table 1 we already mentioned that second member $C1$ becomes reproducible after the building phase. This is not the case for matrix $A1$ that has not been assembled at the interface nodes and so may differ from one subdomain to another. However, $A1$ is build together with its accumulated errors E_{A1} and only appears in the bold matrix-vector product of the conjugate gradient iterations. Hence thanks to the EBE storage, applying the interface node assembly (6) to (10) leads to reproducibility as we describe it now.

The diagonal part MD is associated with its errors E_{MD} so the first term couple is $R_1 = MD\ V$ and $E_{R_1} = E_{MD}\ V$. In another hand, we compute the errors of the finite element assembly of vector $MX_e\ V$ with Relation (8). This yields the second term couple (R_2, E_{R_2}). Hence the (bold) matrix-vector product now is the couple $R = R_1 + R_2$ and $E_R = E_{R_1} + E_{R_2}$ which is assembled on the interface nodes with Relation (6) and finally compensated as $R + E_R$. So all vectors in the conjugate gradient iterations are now reproducible.

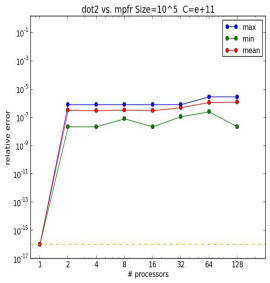
The second source of non reproducibility is the (bracket) dot product which is computed with a parallel dynamic reduction of MPI. In our context, we easily derive a reproducible parallel dot product using existing compensated dot products. Let X and Y be floating point vectors such that, like in Telemac2D, their dot product condition number is reasonably smaller that $1/\mathbf{u}$. Algorithm $dot2$ computes an accurate sequential scalar product [9]. A parallel implementation has to exchange both the partial dot product result and its errors to be reproducible. We illustrate it with some numerical experiments. Figure 2 plots the relative differences compared to a sequential MPFR execution (1000 bits) of three parallel dot products.

Figure 2: Relative differences in dot product algorithms (min, max, mean). X-axis: p processors (1…128). Horizontal dotted line: $\wr\ C \approx 10^{11}$ and $n = 10^5$

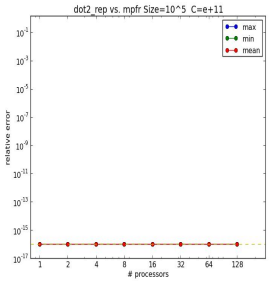
Original Dot:
$r_p = dot(X, Y)$
$r = all\_reduce(r_p)$



Non reproducible Dot2:
$r_p = dot2(X, Y)$
$r = all\_reduce(r_p)$



Reproducible Dot2:
$[r_p, E_{r_p}] = dot2(X, Y)$
$r = all\_reduce(r_p, E_{r_p})$



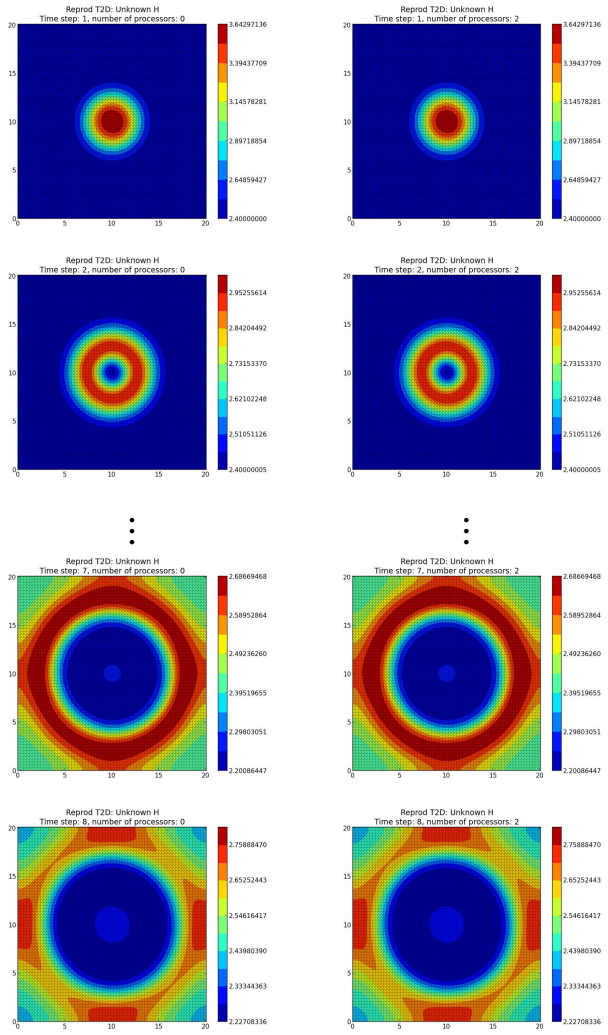
Finally the reproducible versions of the matrix-vector product (bold) and of the dot product (brakets) allows us to perform the conjugate gradient iterations with the same intermediate results and errors (from the divisions and the other operations) in sequential and in parallel. So we recover the reproducibility of the output H as mentioned in the third column of the Table 1.

Reproducibility of the last two steps is now straightforward. The U and V subsystems depend on H. The second members $C2$ and $C3$ are building (from H) and are assembled in the interface node before the resolution, see Table 2. The reproducible interface node assembly applies here during the building phase as described in Section 3.2. Reproducible members $A2$, $C2$, $A3$ and $C3$ leads to reproducible diagonal resolution of the U and V subsystems, as mentioned in the last column of Table 1.

4. Conclusion and future work

We recover the numerical reproducibility of one Telemac2D test case using compensation techniques. Fig-

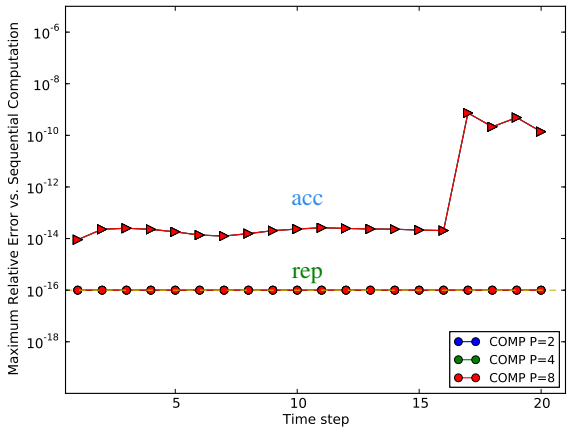
Figure 3: Numerical reproducibility for the water depth in gouttedo



ure 3 now displays the reproducible simulation of the gouttedo test case: no more white plot appears. The same figures also occur while varying the number of processors.

Figure 4 displays two last measures. The rep plot is the maximum relative error over the whole gouttedo domain between the compensated parallel simulation and the compensated sequential. The number of processors varies: $p = 2, 4, 8$. All the plots are superposed and constant at the precision level. This illustrates the reproducibility of the compensated simulations. The second plot acc displays the maximum relative difference between the original sequential Telemac2D simulation and the compensated ones. Relative differences varies from 10^{-14} to 10^{-10}. This validates the compensated simulations that are very similar to the original sequential Telemac2D simulation. As already mentioned, this latter is considered as the reference simulation. Nevertheless since compensation provides more accuracy than one working precision computation, this curve certainly displays the loss of accuracy of the reference result.

Figure 4: Reproducibility of the compensated simulation and accuracy compared to the original Telemac2D for the water depth in gouttedo. X-axis: time steps ($1 \ldots 20 \times 0.2$ sec). Y-axis: maximum relative difference. Number of processors: $p = 2, 4, 8$.



To obtain this reproducible results, it was important to identify the sources where the rounding error differ between the sequential and the parallel simulations. In Telemac2D, the first source comes from a non-deterministic error propagation at the interfaces nodes. So it was sufficient to store these errors and to compensate them after every interface node assembly step. This correction applies for both the parallel and the sequential simulation to yield the expected reproducibility. The second source is the dynamic reduction of the parallel implementation for the dot product used in the conjugate gradient iterations. Compensation yields easy and efficient solution exchanging and reducing both the partial values and the correction ones (the errors). With compensated algorithm, we have been able to obtain the reproducibility of the Telemac2D simulation for the gouttedo test case. The simulation accuracy has been certainly improved. We did not measure any significant time overcost for the whole simulation but a more detailed analysis will be performed.

We will also experiment this approach to some other simulation parameters that define additional physical terms, or introduce preconditioning to the conjugate gradient solving step, or use other linear system solvers, ... We also have to study whether other modes of matrix storage allow us to apply successfully these compensated techniques. Other solutions like integer conversion and reproducible sums [3] that have been applied to another openTelemac module in [8] could also be tested here.

Acknowledgment

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

work.

References

[1] Wei-Fan Chiang, Ganesh Gopalakrishnan, Zvonimir Rakamaric, Dong H. Ahn, and Gregory L. Lee. Determinism and reproducibility in large-scale HPC systems. In *5th Workshop on Determinism and Correctness in Parallel Programming.*, WoDet2013. Washington, USA, ACM, 2013.

[2] Theodorus J. Dekker. A floating-point technique for extending the available precision. *Numer. Math.*, 18:224–242, 1971.

[3] James W. Demmel and Hong Diep Nguyen. Fast reproducible floating-point summation. In *Proc. 21th IEEE Symposium on Computer Arithmetic*. Austin, Texas, USA, 2013.

[4] Yun He and ChrisH.Q. Ding. Using accurate arithmetics to improve numerical reproducibility and stability in parallel applications. *J. Supercomput.*, 18:259–277, 2001.

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

[6] William Kahan. Further remarks on reducing truncation errors. *Communications of the ACM*, 8(1):40, 1965.

[7] Donald E. Knuth. *The Art of Computer Programming, Volume 2, Seminumerical Algorithms*. Addison-Wesley, Reading, MA, USA, third edition, 1998.

[8] Philippe Langlois, Rafife Nheili, and Christophe Denis. Numerical Reproducibility: Feasibility Issues. In *NTMS'2015: 7th IFIP International Conference on New Technologies, Mobility and Security*, pages 1–5, Paris, France, July 2015. IEEE, IEEE COMSOC & IFIP TC6.5 WG.

[9] Takeshi Ogita, Siegfried M. Rump, and Shin'ichi Oishi. Accurate sum and dot product. *SIAM J. Sci. Comput.*, 26(6):1955–1988, 2005.

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

[11] Robert W. Robey, Jonathan M. Robey, and Rob Aulwes. In search of numerical consistency in parallel programming. *Parallel Comput.*, 37(4-5):217–229, 2011.

[12] Michela Taufer, Omar Padron, Philip Saponaro, and Sandeep Patel. Improving numerical reproducibility and stability in large-scale numerical simulations on gpus. In *IPDPS*, pages 1–9. IEEE, 2010.

[13] Oreste Villa, Daniel G. Chavarría-Miranda, Vidhya Gurumoorthi, Andrès Márquez, and Sriram Krishnamoorthy. Effects of floating-point non-associativity on numerical computations on massively multithreaded systems. In *CUG 2009 Proceedings*, pages 1–11, 2009.