# Estimation of numerical reproducibility on CPU and GPU

Fabienne Jézéquel[1,2,3], Jean-Luc Lamotte[1,2], and Issam Saïd[1,2]

[1]Sorbonne Universités, UPMC Univ Paris 06, UMR 7606, LIP6, F-75005, Paris, France
[2]CNRS, UMR 7606, LIP6, F-75005, Paris, France
[3]Université Panthéon-Assas, 12 place du Panthéon, F-75231 Paris CEDEX 05, France
*{Fabienne.Jezequel,Jean-Luc.Lamotte,Issam.Said}@lip6.fr*

***Abstract*—Differences in simulation results may be observed from one architecture to another or even inside the same architecture. Such reproducibility failures are often due to different rounding errors generated by different orders in the sequence of arithmetic operations. Reproducibility problems are particularly noticeable on new computing architectures such as multicore processors or GPUs (Graphics Processing Units). DSA (Discrete Stochastic Arithmetic) enables one to estimate rounding error propagation in simulation programs. In this paper, it is shown that DSA can be used to estimate which digits in simulation results may be different from one environment to another because of rounding errors. A particular implementation of DSA, which enables numerical validation in hybrid CPU-GPU environments, is described. The estimation of numerical reproducibility using DSA is illustrated by a wave propagation code which can be affected by reproducibility problems when executed on different architectures.**

## I. Introduction

Results of numerical simulations may be different from one architecture to another, or even inside the same architecture if they are computed using different compilers for instance. In sequential or parallel environments, different orders in the sequence of floating-point operations may lead to differences in rounding error propagation and therefore to reproducibility failures. It must be pointed out that the cause of differences in results may be difficult to identify: rounding errors or bug? Such differences are particularly noticeable with new computing architectures such as multicore processors, GPUs (Graphics Processing Units) and APUs (Accelerated Processing Units). In high performance numerical simulations, reproducibility problems have been identified in various domains: energy science [1], climate science [2], atomic or molecular dynamics [3], [4], fluid dynamics [5]. Various studies have been carried out on numerical reproducibility on different architectures. On the one hand, strategies have been proposed [2], [3], [4], [5] to improve numerical accuracy, using for instance accurate summations. Other works aim at forcing the reproducibility of results, either affected by the same rounding errors [6], [7] or correctly rounded [8], [9], [10], [11].

DSA (Discrete Stochastic Arithmetic) [12], [13] enables one to estimate rounding error propagation in simulation programs. This paper shows that DSA can be used, not to force a code to be reproducible, but to estimate the number of digits in the results which may be different from one execution to another because of rounding errors. The CADNA[1] library [14], [15], [16], which implements DSA, enables the numerical quality estimation of sequential programs in C or Fortran and of parallel programs using MPI for communication [17]. This paper describes a version of CADNA, briefly introduced in [18], that can be used in hybrid CPU-GPU environments to estimate rounding errors in CUDA programs. This paper is organized as follows. In Section 2, differences in results provided by a wave propagation code executed on several architectures - CPU, GPU and APU - are pointed out. Section 3 presents the principles of DSA. Section 3 also describes the CADNA library and presents the particularities of a CADNA version for CPU-GPU codes. Section 4 shows that the reproducibility problems observed in wave propagation results can be explained by rounding error propagation thanks to the CADNA library. Finally, concluding remarks are presented in Section 5.

## II. Reproducibility failures in a wave propagation code

We consider the three-dimensional acoustic wave equation

$$\frac{1}{c^2}\frac{\partial^2 u}{\partial t^2} - \sum_{b\in\{x,y,z\}} \frac{\partial^2}{\partial b^2} u = 0,$$

where $u$ is the particle velocity, $c$ is the wave velocity, and $t$ is the time. This equation, used for instance in oil exploration [19], is solved with an explicit finite difference scheme of order 2 in time and $p$ in space (in our case $p = 8$). We denote by $u^n_{i,j,k}$ (respectively $f^n_{i,j,k}$) the wave (respectively source) field in $(i, j, k)$ coordinates and $n$th time step, $a_l$ ($l = -p/2, \ldots, p/2$) the finite difference coefficients, $\Delta t$ the time step and $\Delta h$ the spatial step size. Two mathematically equivalent implementations of the finite difference scheme are proposed:

$$u^{n+1}_{i,j,k} = 2u^n_{i,j,k} - u^{n-1}_{i,j,k} + \frac{c^2\Delta t^2}{\Delta h^2} S^n_{i,j,k} + c^2\Delta t^2 f^n_{i,j,k}$$

[1]URL address: http://www.lip6.fr/cadna

where

$$S^n_{i,j,k} = \sum_{l=-p/2}^{p/2} a_l \left( u^n_{i+l,j,k} + u^n_{i,j+l,k} + u^n_{i,j,k+l} \right) \quad (1)$$

or

$$S^n_{i,j,k} = \sum_{l=-p/2}^{p/2} a_l u^n_{i+l,j,k} + \sum_{l=-p/2}^{p/2} a_l u^n_{i,j+l,k} + \sum_{l=-p/2}^{p/2} a_l u^n_{i,j,k+l}. \quad (2)$$

In order to satisfy the CFL (Courant-Friedrichs-Lewy) necessary stability condition [20], the time step is computed by taking into account the wave velocity $c$, the spatial step size $\Delta h$ and the spatial order $p$. Because these two implementations require the same number of arithmetic operations, they should lead to similar performance. However it would be interesting to determine whether they differ in the numerical quality of their results.

The code is executed for $64 \times 64 \times 64$ space steps and 1000 time iterations in IEEE-754 binary32 arithmetic with rounding to the nearest [21] and the following environments:

- AMD Opteron 6168 CPU with gcc 4.7.2 compiler;
- NVIDIA C2050 GPU (Graphics Processing Unit) with CUDA (Compute Unified Device Architecture) platform;
- NVIDIA K20c GPU with OpenCL (Open Computing Language);
- AMD Radeon HD 7970 GPU with OpenCL;
- AMD Trinity APU (Accelerated Processing Unit) with OpenCL.

Different kinds of reproducibility problems are observed. The results numerically vary

1) from one execution to another inside a GPU or an APU; these repeatability problems are due to differences in the execution order of the threads;
2) from one implementation of the finite difference scheme to another; the maximal relative difference between results is of the order of $10^{-1}$ to 1 depending on the architecture, and the mean value of the relative difference between results is of the order of $10^{-5}$ whatever the architecture;
3) from one architecture to another; again, the maximal relative difference between results is of the order of $10^{-1}$ to 1 and its mean value is $10^{-5}$.

Indeed if two sets of results computed in binary32 are compared, the results at the same space coordinates can have from 0 to 7 significant digits in common, and the average number of common significant digits is about 4. We recall that results computed using binary32 arithmetic precision can have at most 7 correct significant digits. To illustrate these reproducibility problems, Table I presents at three space coordinates $(i, j, k)$, $0 \leq i, j, k \leq 63$, the results obtained after 1000 times iterations using the processing units and languages previously mentioned. These results have different orders of magnitude. Both implementations of the finite difference scheme are considered. Considering the example points presented in Table I, any two results computed at the same point in the space domain have 3 to 6 common significant digits.

TABLE I
RESULTS COMPUTED AT THREE DIFFERENT POINTS IN THE SPACE DOMAIN

| | Point in the space domain | | |
|---|---|---|---|
| | $p_1$: (0, 19, 62) | $p_2$: (50, 12, 2) | $p_3$: (20, 1, 46) |
| AMD Opteron CPU with gcc | | | |
| scheme 1 | -1.110479 | 54.54238 | 614.1038 |
| scheme 2 | -1.110426 | 54.54199 | 614.1035 |
| NVIDIA C2050 GPU with CUDA | | | |
| scheme 1 | -1.110204 | 54.54224 | 614.1046 |
| scheme 2 | -1.109869 | 54.54244 | 614.1047 |
| NVIDIA K20c GPU with OpenCL | | | |
| scheme 1 | -1.109953 | 54.54218 | 614.1044 |
| scheme 2 | -1.111517 | 54.54185 | 614.1024 |
| AMD Radeon GPU with OpenCL | | | |
| scheme 1 | -1.109940 | 54.54317 | 614.1038 |
| scheme 2 | -1.110111 | 54.54170 | 614.1044 |
| AMD Trinity APU with OpenCL | | | |
| scheme 1 | -1.110023 | 54.54169 | 614.1062 |
| scheme 2 | -1.110113 | 54.54261 | 614.1049 |

## III. ESTIMATING ROUNDING ERRORS WITH DISCRETE STOCHASTIC ARITHMETIC (DSA)

### A. Principles of DSA

Based on a probabilistic approach, the CESTAC method [12] allows the estimation of rounding error propagation which occurs with floating-point arithmetic. When no overflow occurs, the exact result of any non exact floating-point arithmetic operation is bounded by two consecutive floating-point values $R^-$ and $R^+$. The basic idea of the method is to perform each arithmetic operation $N$ times, randomly rounding each time, with a probability of 0.5, to $R^-$ or $R^+$. The computer's deterministic arithmetic, therefore, is replaced by a stochastic arithmetic where each arithmetic operation is performed $N$ times before the next one is executed, thereby propagating the rounding error differently each time. The CESTAC method furnishes us with $N$ samples $R_1, \ldots, R_N$ of the computed result $R$. The value of the computed result, $\overline{R}$, is the mean value of $\{R_i\}_{1 \leqslant i \leqslant N}$ and the number of exact significant digits in $\overline{R}$, $C_{\overline{R}}$, is estimated as

$$C_{\overline{R}} = \log_{10} \left( \frac{\sqrt{N}\,|\overline{R}|}{\sigma \tau_\beta} \right)$$

$$\text{with } \overline{\mathrm{R}} = \frac{1}{\mathrm{N}} \sum_{\mathrm{i}=1}^{\mathrm{N}} \mathrm{R_i} \text{ and } \sigma^2 = \frac{1}{\mathrm{N}-1} \sum_{\mathrm{i}=1}^{\mathrm{N}} \left( \mathrm{R_i} - \overline{\mathrm{R}} \right)^2.$$

$\tau_\beta$ is the value of Student's distribution for $N-1$ degrees of freedom and a probability level $1-\beta$. In practice $N = 3$ and $\beta = 0.05$. Indeed, it has been shown [22], [23] that $N = 3$ is in some reasonable sense the optimal value. The estimation with $N = 3$ is more reliable than with $N = 2$ and increasing the size of the sample does not improve the quality of the estimation. The probability of overestimating

the number of exact significant digits of at least 1 is 0.054% and the probability of underestimating the number of exact significant digits of at least 1 is 29%. By choosing $\beta = 0.05$, we prefer to guarantee a minimal number of exact significant digits with a high probability (99.946%), even if we are often pessimistic by 1 digit. The complete theory can be found in [12], [23].

The validity of $C_{\overline{R}}$ is compromised if the two operands in a multiplication or the divisor in a division are not significant [23]. It is essential, therefore, that these results with no significance are detected and reported, since their subsequent use may invalidate the method. The need for this dynamic control of multiplications and divisions has led to the concept of the computational zero. A computed result is a computational zero, denoted by @.0, if $\forall i, R_i = 0$ or $C_{\overline{R}} \leq 0$. This means that a computational zero is either a mathematical zero or a number without any significance, *i.e.* numerical noise.

To establish consistency between the arithmetic operators and the relational operators, discrete stochastic relations are defined as follows. Let $X = (X_1, ..., X_N)$ and $Y = (Y_1, ..., Y_N)$ be two results computed using the CESTAC method, we have from [24]

$X = Y$ if and only if $X - Y = @.0$;

$X > Y$ if and only if $\overline{X} > \overline{Y}$ and $\mathrm{X} - \mathrm{Y} \neq @.0$;

$X \geq Y$ if and only if $\overline{X} \geq \overline{Y}$ or $\mathrm{X} - \mathrm{Y} = @.0$.

Discrete Stochastic Arithmetic (DSA) [12], [13] is the combination of the CESTAC method, the concept of the computational zero, and the discrete stochastic relationships.

### *B. Numerical validation of sequential codes using DSA*

The CADNA software [14], [15], [16] is a library which implements DSA in any code written in C++ or in Fortran and allows to use new numerical types: the stochastic types. In practice, classic floating-point variables are replaced by the corresponding stochastic variables, which are composed of three floating-point values and an integer to store the accuracy. The library contains the definition of all arithmetic operations and order relations for the stochastic types. For instance, let us consider an arithmetic operation $\circ \in \{+, -, *, /\}$ between two stochastic variables A and B. This arithmetic operation is performed three times on the associated floating-point values $A_i \circ B_i$, the rounding mode being randomly set to rounding towards $+\infty$ or $-\infty$. The control of accuracy is performed only on variables of stochastic type. Only exact significant digits of a stochastic variable are printed or "@.0" for a computational zero. Because all operators are redefined for stochastic variables, the use of CADNA in a program requires only a few modifications: essentially changes in the declarations of variables and in input/output statements. The CADNA software has been successfully used for the numerical validation of real-life applications [17], [25], [26], [27], [28].

Attention has been paid to rounding mode setting in terms of performance, because the rounding mode must be frequently changed. On CPU the rounding mode is determined by two bits in the Control Word, a 16-bit register in the FPU (Floating-Point Unit). At the beginning of a program using CADNA, the rounding mode is arbitrarily set to $-\infty$. Then the rounding mode is randomly changed using the CADNA *rnd_switch* function that switches the rounding mode from $+\infty$ to $-\infty$, or from $-\infty$ to $+\infty$. To reduce the cost of rounding mode changes, the *rnd_switch* function is written in assembly language and is specific to the processor and the compiler chosen. The *rnd_switch* function changes in the FPU Control Word the two bits associated with the rounding mode. For performance reasons, a random number generator is not called at each arithmetic operation. A long random sequence is generated at the beginning of the program and stored in an array. Then successive array elements are used cyclically when random numbers are required.

CADNA can detect numerical instabilities which occur during the execution of the code. When a numerical instability is detected, dedicated CADNA counters are incremented. At the end of the run, the value of these counters together with appropriate warning messages are printed on standard output. These warnings are of two types.

1) Warnings related to the self-validation of CADNA. These include: unstable multiplication where the two operands are computational zeroes and unstable division where the divisor is a computational zero. These warnings indicate that the validity of $C_{\overline{R}}$ has been compromised and the CADNA results cannot be relied on.
2) Warnings concerning other numerical instabilities. These instabilities can occur in overloaded mathematical functions or in branching statements involving a computational zero. A numerical instability is also reported in the case of a cancellation, *i.e.* the subtraction of two very close values which generates a sudden loss of accuracy.

At the end of the run, each type of anomaly together with their occurrences are printed. If no anomaly has been detected the computed results are reliable and the accuracy of each has been correctly estimated up to a certain probability. Otherwise the messages need to be analysed, the source of the anomaly identified and, if necessary, the code changed. The user can specify the instabilities to be detected. One may choose, for instance, to activate only self-validation, to detect all types of instabilities or to deactivate the detection of instabilities.

### *C. Numerical validation of hybrid CPU-GPU codes using DSA*

An asynchronous implementation of the CESTAC method for the estimation of rounding errors in GPU codes written in CUDA is proposed in [29]. Unfortunately with such an implementation of the CESTAC method, the whole code is executed several times with the random rounding mode and no instability in arithmetic operations can be detected. We present here a version of CADNA which implements DSA for the numerical validation of hybrid CPU-GPU codes written in

CUDA. This version differs from the sequential version in two main respects: rounding mode change and instability detection.

On GPU an arithmetic operation can be performed with a specified rounding mode. For instance a multiplication with rounding towards $+\infty$ can be executed using the *fmul_ru* function and a multiplication with rounding towards $-\infty$ using the *fmul_rd* function. Therefore a stochastic operation on GPU implies three floating-point operations randomly rounded towards $+\infty$ or $-\infty$ using the appropriate arithmetic function. Unlike on CPU, random numbers are not stored in a global array on GPU, because it is incompatible with GPU programming paradigms. Each GPU thread being independent, it generates random numbers using its own seed and taking into account its own thread index. On GPU, random numbers are generated when they are required, *i.e.* during the stochastic operations. As a remark, because of the robustness of accuracy estimation by DSA [22], the quality of the random number generator is not a critical issue: only boolean values are required.

On CPU, numerical instabilities that occur during the execution are counted. Such a count is not performed on GPU because it would consume shared memory and require many atomic operations. On GPU an unsigned char is associated with each result to store the numerical instabilities that have affected it. Each bit of this char is associated with a type of instability. For instance, its last bit is set to 1 if the result has been affected by at least one unstable multiplication. Therefore in the CPU-GPU version of CADNA a stochastic variable is composed of three floating-point variables and four unsigned char: one for the accuracy, one for the instabilities and two for padding to respect memory alignment. Instability detection increases the execution time with CADNA. Cancellation detection is particularly costful because it requires to compare for all additions and subtractions the operands accuracy with the result accuracy. For performance reasons, the main instabilities can be detected with the GPU version of CADNA: the instabilities related to the self-validation of CADNA (unstable multiplications and unstable divisions) and the unstable branching statements.

## IV. Estimation of reproducibility in wave propagation results by means of DSA

The acoustic wave propagation code has been executed with the CADNA library on CPU and on GPU. Results presented in this section have been computed in the following environments:

- an AMD Opteron 6168 CPU with gcc 4.7.2 compiler;
- an NVIDIA C2050 GPU with CUDA 5.0 platform.

With implementations (1) and (2) of the finite difference scheme, the number of exact significant digits in the results computed with CADNA varies from 0 to 7. On CPU its mean value is 4.06 with both schemes; on GPU it is 3.43 with scheme (1) and 3.49 with scheme (2). These remarks are consistent with the observations described in Section II. Numerical instabilities occur during the execution: 272,394 losses of accuracy due to cancellations with scheme (1) and 285,186 with scheme (2). This kind of instability is detected by the CPU version of CADNA if the subtraction of two close floating-point numbers leads to a loss of accuracy of at least 4 digits.

Table II presents results obtained on CPU and on GPU at the same points in the space domain as in Table I. These results have been computed, on the one hand, using CADNA and, on the other hand, using IEEE floating-point arithmetic with rounding to the nearest. With CADNA, only the exact significant digits, *i.e.* the digits not affected by rounding errors, are printed. Results in the first four rows in Table II have been computed using binary32 arithmetic precision. Results in the last row have been obtained in binary64 on CPU with CADNA. Although CADNA prints 11 to 14 exact significant digits in these three results, only their first 10 digits are reported in Table II. The number of exact significant digits estimated by CADNA depends on the point considered in the space domain. As already mentioned in Section III, accuracy estimation by DSA is rather pessimistic than optimistic. Because of the probabilistic aspect of DSA, the number of exact significant digits estimated by CADNA may slightly differ on CPU and on GPU. In Table II one can notice that the digits provided by CADNA in binary32 are in common with those computed in binary64. Results reported in Table II have been computed using implementation (1) of the finite difference scheme. The same digits are provided by CADNA with the other implementation, except one less digit is given on GPU for point $p_1$.

TABLE II
Results computed at three different points in the space domain with and without CADNA using implementation (1) of the finite difference scheme

| | Point in the space domain | | |
|---|---|---|---|
| | $p_1$: (0, 19, 62) | $p_2$: (50, 12, 2) | $p_3$: (20, 1, 46) |
| IEEE CPU | -1.110479 | 54.54238 | 614.1038 |
| IEEE GPU | -1.110204 | 54.54224 | 614.1046 |
| CADNA CPU | -1.1 | 54.54 | 614.104 |
| CADNA GPU | -1.11 | 54.5 | 614.10 |
| Reference | -1.108603879 | 54.54034021 | 614.1041156 |

Figures 1 and 2 present, respectively on CPU and on GPU, the number of exact significant digits estimated by CADNA in the results computed with scheme (1) with respect to their absolute values. Similar results are observed with the other scheme. The highest results (in absolute value) are affected by low rounding errors and the highest rounding errors impact negligible results. Although the same trend can be observed on CPU and on GPU, there are differences between the two distributions due to the probabilistic aspect of DSA. Depending on the point in the space domain, the number of exact significant digits may be higher on CPU or on GPU. The average difference between results accuracy on CPU and on GPU is 0.6 digit. Furthermore because of differences in the execution order of the threads, the accuracy distribution may be slightly different from one execution to another on GPU.

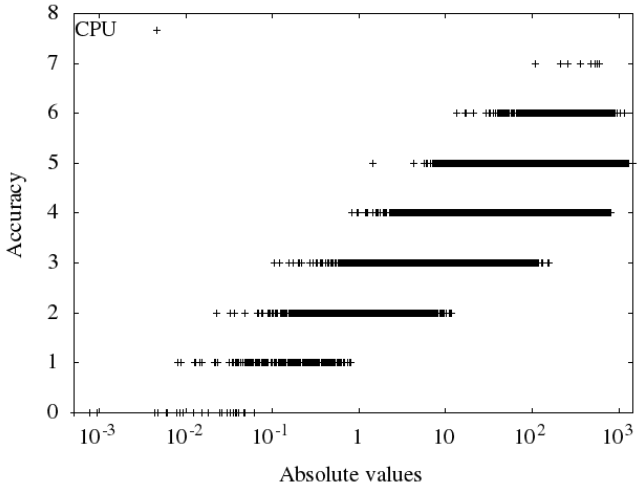


Fig. 1. Number of exact significant digits in the results computed on CPU with respect to their absolute values.

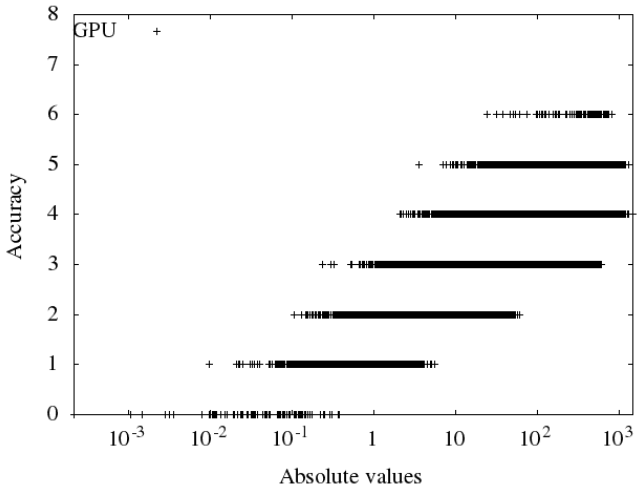


Fig. 2. Number of exact significant digits in the results computed on GPU with respect to their absolute values.

Table III presents execution times of the acoustic wave propagation code in the environments mentioned at the beginning of this section. Because the execution times measured with implementations (1) and (2) of the finite difference scheme are similar, only the performance of implementation (1) is reported in Table III. The code has been run in binary32 both on CPU and on GPU. CADNA has been used on CPU with several kinds of instability detection:

- the detection of all kinds of instabilities;
- no detection of instabilities. With this mode, which is not recommended, the execution time can be considered the minimum that can be obtained whatever instability detection chosen;
- the detection of unstable multiplications, unstable divisions and unstable branching statements. This mode, which enables the self-validation of CADNA, is also available on GPU.

One can notice that the cost of CADNA with instability detection in multiplications, divisions and branching statements is very close to its cost with no instability detection. Actually this code cannot generate such instabilities: in all multiplications at least one operand is a constant, all divisors are constants and it has no branching statement. The cost of CADNA with the detection of any kind of instability is 2.6 times higher. This is essentially due to the cancellation detection which is particularly expensive in terms of execution time. The cost of CADNA on GPU is about twice lower than on CPU with the same level of instability detection. This may be explained by the pipeline flush at each change of rounding mode on CPU which affects instruction level parallelism.

TABLE III
EXECUTION TIMES WITH AND WITHOUT CADNA ON CPU AND GPU

| CPU | | | |
|---|---|---|---|
| execution | instability detection | execution time (s) | ratio |
| IEEE | - | 110.8 | 1 |
| CADNA | all instabilities | 4349 | 39.3 |
| | no instability | 1655 | 14.9 |
| | mul., div., branching | 1663 | 15.0 |
| **GPU** | | | |
| execution | instability detection | execution time (s) | ratio |
| IEEE | - | 0.80 | 1 |
| CADNA | mul., div., branching | 5.73 | 7.2 |

## V. CONCLUSION

In this paper, we have shown that DSA can provide an estimation of the reproducibility of numerical programs. By estimating which digits are affected by rounding errors, DSA may explain why differences are observed in the results of a program executed in different environments. Therefore when the deployment of a code on a parallel architecture generates differences in the computed results, the presence of a bug can possibly be discarded. Based on a probabilistic approach, DSA can provide a trend of the distribution or the evolution of results accuracy. If results differences are due to different orders in the sequence of floating-point operations, a similar trend should be provided by DSA whatever the environment chosen. But a sequential implementation of DSA is not sufficient and efficient methods must be proposed for the numerical validation of large scale simulation programs. This paper has shown the feasibility of numerical validation with DSA for CPU-GPU programs. Furthermore DSA can be used for accuracy estimation in distributed memory environments [17]. It has recently been shown how to take advantage of SIMD units such as AVX (Advanced Vector eXtensions) in programs using DSA [30]. However, work must still be carried out to extend efficiently DSA to emerging computing architectures that are prone to numerical reproducibility failures.

## REFERENCES

[1] O. Villa, D. Chavarría-Miranda, V. Gurumoorthi, A. Márquez, and S. Krishnamoorthy, "Effects of floating-point non-associativity on numerical computations on massively multithreaded systems," in *Cray User Group Meeting (CUG 2009)*, Atlanta, Georgia, USA, May 2009, pp. 1–11.

[2] Y. He and C. Ding, "Using accurate arithmetics to improve numerical reproducibility and stability in parallel applications," *The Journal of Supercomputing*, vol. 18, no. 3, pp. 259–277, 2001.

[3] M. Cleveland, T. Brunner, N. Gentile, and J. Keasler, "Obtaining identical results with double precision global accuracy on different numbers of processors in parallel particle Monte Carlo simulations," *Journal of Computational Physics*, vol. 251, pp. 223–236, 2013.

[4] M. Taufer, O. Padron, P. Saponaro, and S. Patel, "Improving numerical reproducibility and stability in large-scale numerical simulations on GPUs," in *IEEE International Symposium on Parallel Distributed Processing (IPDPS)*, Atlanta, Georgia, USA, 2010, pp. 1–9.

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

[6] J. Demmel and H. D. Nguyen, "Fast reproducible floating-point summation," in *21st IEEE Symposium on Computer Arithmetic (ARITH 21)*, Austin, Texas, USA, Apr. 2013, pp. 163–172.

[7] ——, "Parallel reproducible summation," in *IEEE Transactions on Computers, Special Section on Computer Arithmetic*, to appear.

[8] S. Collange, D. Defour, S. Graillat, and R. Iakymchuk, "A reproducible accurate summation algorithm for High-Performance Computing," in *SIAM Workshop on Exascale Applied Mathematics Challenges and Opportunities (EX14) held as part of the 2014 SIAM Annual Meeting*, Chicago, Illinois, USA, Jul. 2014.

[9] ——, "Reproducible and accurate matrix multiplication for High-Performance Computing," in *The 16th GAMM-IMACS International Symposium on Scientific Computing, Computer Arithmetic and Validated Numerics (SCAN'14)*, Würzburg, Germany, Sep. 2014.

[10] R. Iakymchuk, S. Collange, D. Defour, and S. Graillat, "Reproducible triangular solvers for high-performance computing," in *12th International Conference on Information Technology: New Generations (ITNG)*, Las Vegas, Nevada, USA, Apr. 2015. [Online]. Available: http://hal.archives-ouvertes.fr/hal-01116588v2

[11] S. Collange, D. Defour, S. Graillat, and R. Iakymchuk, "Full-speed deterministic bit-accurate parallel floating-point summation on multi- and many-core architectures," http://hal.archives-ouvertes.fr/hal-00949355v3/PDF/superaccumulator.pdf, Feb. 2015. [Online]. Available: http://hal.archives-ouvertes.fr/hal-00949355v3

[12] J. Vignes, "A stochastic arithmetic for reliable scientific computation," *Mathematics and Computers in Simulation*, vol. 35, pp. 233–261, 1993.

[13] ——, "Discrete Stochastic Arithmetic for validating results of numerical software," *Numerical Algorithms*, vol. 37, no. 1–4, pp. 377–390, Dec. 2004.

[14] F. Jézéquel and J.-M. Chesneaux, "CADNA: a library for estimating round-off error propagation," *Computer Physics Communications*, vol. 178, no. 12, pp. 933–955, 2008.

[15] F. Jézéquel, J.-M. Chesneaux, and J.-L. Lamotte, "A new version of the CADNA library for estimating round-off error propagation in Fortran programs," *Computer Physics Communications*, vol. 181, no. 11, pp. 1927–1928, 2010.

[16] J.-L. Lamotte, J.-M. Chesneaux, and F. Jézéquel, "CADNA_C: A version of CADNA for use with C or C++ programs," *Computer Physics Communications*, vol. 181, no. 11, pp. 1925–1926, 2010.

[17] S. Montan and C. Denis, "Numerical verification of industrial numerical codes," in *ESAIM: Proc.*, vol. 35, Mar. 2012. doi: 10.1051/proc/201235006 pp. 107–113.

[18] F. Jézéquel and J.-L. Lamotte, "Numerical validation of Slater integrals computation on GPU," in *The 14th GAMM-IMACS International Symposium on Scientific Computing, Computer Arithmetic and Validated Numerics (SCAN'10)*, Lyon, France, Sep. 2010, pp. 78–79.

[19] H. Calandra, R. Dolbeau, P. Fortin, J.-L. Lamotte, and I. Said, "Forward seismic modeling on AMD Accelerated Processing Unit," in *Rice Oil & Gas HPC Workshop*, Houston, Texas, USA, Mar. 2013.

[20] B. Gustafsson, H.-O. Kreiss, and J. Oliger, *Time Dependent Problems and Difference Methods*, 2nd ed. Wiley, 2013.

[21] IEEE Computer Society, *IEEE Standard for Floating-Point Arithmetic*. IEEE Standard 754-2008, Aug. 2008. ISBN 978-0-7381-5752-8

[22] J.-M. Chesneaux and J. Vignes, "Sur la robustesse de la méthode CESTAC," *Comptes Rendus de l'Académie des Sciences - Series I - Mathematics*, vol. 307, pp. 855–860, 1988.

[23] J.-M. Chesneaux, "L'arithmétique stochastique et le logiciel CADNA," Habilitation à diriger des recherches, Université Pierre et Marie Curie, Paris, France, Nov. 1995.

[24] J.-M. Chesneaux and J. Vignes, "Les fondements de l'arithmétique stochastique," *Comptes Rendus de l'Académie des Sciences - Series I - Mathematics*, vol. 315, pp. 1435–1440, 1992.

[25] F. Jézéquel, F. Rico, J.-M. Chesneaux, and M. Charikhi, "Reliable computation of a multiple integral involved in the neutron star theory," *Math. Comput. Simulation*, vol. 71, no. 1, pp. 44–61, 2006.

[26] N. Scott, F. Jézéquel, C. Denis, and J.-M. Chesneaux, "Numerical 'health check' for scientific codes: the CADNA approach," *Computer Physics Communications*, vol. 176, no. 8, pp. 507–521, Apr. 2007.

[27] F. Jézéquel, J.-L. Lamotte, and O. Chubach, "Parallelization of Discrete Stochastic Arithmetic on multicore architectures," in *10th International Conference on Information Technology: New Generations (ITNG), Las Vegas, Nevada (USA)*, Apr. 2013.

[28] J. Brajard, P. Li, F. Jézéquel, H.-S. Benavidès, and S. Thiria, "Numerical Validation of Data Assimilation Codes Generated by the YAO Software," in *SIAM Annual Meeting, San Diego, California (USA)*, Jul. 2013.

[29] W. Li, S. Simon, and S. Kiess, "On the numerical sensitivity of computer simulations on hybrid and parallel computing systems," in *International Conference on High Performance Computing and Simulation (HPCS)*, Istanbul, Turkey, Jul. 2011, pp. 510–516.

[30] P. Eberhart, J. Brajard, P. Fortin, and F. Jézéquel, "Towards high performance stochastic arithmetic," in *The 16th GAMM-IMACS International Symposium on Scientific Computing, Computer Arithmetic and Validated Numerics (SCAN'14)*, Würzburg, Germany, Sep. 2014.