Parallelization of Discrete Stochastic Arithmetic on multicore architectures

F. Jézéquel†, J.-L. Lamotte† and O. Chubach*
†UPMC Univ Paris 06, UMR 7606, Laboratoire d'Informatique de Paris 6,
4 place Jussieu, 75252 Paris CEDEX 05, France
Email: {Fabienne.Jezequel,Jean-Luc.Lamotte}@lip6.fr
*Odessa I.I.Mechnikov National University,
Dvoryans'ka str. 2, Odessa, 65082, Ukraine
Email: elenachubach@gmail.com

***Abstract*—Discrete Stochastic Arithmetic (DSA) estimates round-off error propagation in a program. It is based on a synchronous execution of several instances of the program to control using a random rounding mode. In this paper we show how we can take advantage of multicore processors, which are nowadays widespread, to reduce the cost of DSA in terms of execution time. Several processes execute in parallel different instances of the program and exchange data when necessary. Several strategies are compared for the estimation of the result accuracy and the detection of numerical instabilities. With our parallel implementation, the cost of DSA is reduced by a factor of about 2 compared with the sequential approach. Our parallel implementation of DSA has been used successfully for the numerical validation of a real-life application.**

***Index Terms*—Discrete Stochastic Arithmetic, floating-point arithmetic, multicore processors, numerical validation, round-off errors;**

I. Introduction

The increasing power of current computers enables one to solve more and more complex problems. Then it is necessary to perform more and more floating-point operations, each one leading to a round-off error. Several approaches exist for the round-off error analysis. Backward error analysis [1], [2] provides error bounds at a moderate computational cost. But this approach does not apply to any kind of problem and may provide pessimistic bounds. Interval arithmetic [3], [4] computes guaranteed error bounds. Because classical numerical algorithms may lead to pessimistic bounds with interval arithmetic, specific algorithms exist for solving problems with interval arithmetic. Discrete Stochastic Arithmetic (DSA) [5], [6] computes estimations of round-off errors. DSA is based on the synchronous execution of N instances of a program using a random rounding mode. Numerical validation with DSA requires only a few modifications in the program to control. DSA has been successfully used for the numerical validation of real-life applications [7], [8].

Unfortunately the multiple executions inherent to DSA lead to a problematic computational cost. In this paper we show how multicore architectures, which are nowadays widespread, can reduce this cost. Our aim is to perform in parallel on a multicore system the N executions of a program required by DSA. Because the program to control is sequential, this approach is very different from the numerical validation using DSA of codes running on distributed memory architectures [9], [10] or on CPU-GPU hybrid systems [11], [12].

Several parallel implementations of DSA have been proposed [13]–[15]. However their data exchanges, based on the MPI communication library, require many data copies and are very time consuming. These implementations have been compared in terms of performance on a multi-processor shared memory computer with an implementation using UNIX functions for data exchange, in favour of the latter. Recently a parallel implementation of DSA based on OpenMP has been presented [16]. It has been used on simple Fortran programs for performance benchmarking, but not yet to control large scientific codes.

Because numerical simulation programs are usually executed on UNIX-based operating systems, the parallel implementations of DSA presented in this paper use UNIX functions to create processes and manage data exchange between them. Several strategies are compared for the detection of numerical instabilities which may occur during the execution. This paper is organized as follows. Sect. II exposes the principles of DSA. Sect. III presents the features of the sequential version of the CADNA[1] software [17]–[19] which implements DSA. Sect. IV describes different strategies for the parallelization of DSA on multicore systems. These strategies are compared in terms of performance in Sect. V. Sect. VI shows how our parallel implementations of DSA have been used for the numerical validation of a real-life application. Finally, in Sect. VI concluding remarks are presented.

II. Discrete Stochastic Arithmetic

A. Principles of the CESTAC method

Based on a probabilistic approach, the CESTAC method [5] allows the estimation of round-off error propagation which occurs with floating-point arithmetic. When no overflow occurs, the exact result, r, 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

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

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 round-off error differently each time.

It has been proved [20] that the computed result R of n elementary arithmetic operations is modelled to the first order in 2^{-p} as:

$$R \approx Z = r + \sum_{i=1}^{n} g_i(d) 2^{-p} z_i \tag{1}$$

where r is the exact result, $g_i(d)$ are coefficients depending exclusively on the data and on the code, p is the number of bits in the mantissa and z_i are independent uniformly distributed random variables on $[-1, 1]$.

From Eq. 1, we deduce that:

1) the mean value of the random variable Z is the exact result r,
2) the distribution of Z is a quasi-Gaussian distribution.

Then by identifying R and Z, *i.e.* by neglecting all the second order terms, Student's test can be used to estimate the accuracy of R. From N samples R_i, $i = 1, 2, ..., N$,

$$\forall \beta \in [0,1], \exists \tau_\beta \in \mathbb{R} \text{ s.t. } P\left(\left| \overline{R} - r \right| \leq \frac{\sigma \tau_\beta}{\sqrt{N}} \right) = 1 - \beta \tag{2}$$

where

$$\overline{R} = \frac{1}{N} \sum_{i=1}^{N} R_i \quad \text{and} \quad \sigma^2 = \frac{1}{N-1} \sum_{i=1}^{N} \left(R_i - \overline{R} \right)^2 . \tag{3}$$

τ_β is the value of Student's distribution for $N-1$ degrees of freedom and a probability level $1 - \beta$.

Therefore the number of decimal significant digits common to $\overline{R}$ and r can be estimated with the following equation.

$$C_{\overline{R}} = \log_{10} \left(\frac{\sqrt{N} \left| \overline{R} \right|}{\sigma \tau_\beta} \right) . \tag{4}$$

Thus the implementation of the CESTAC method in a code providing a result R consists in:

- performing N times this code with the random rounding mode, which is obtained by using randomly the upward or downward rounding mode; we then obtain N samples R_i of R
- choosing as the computed result the mean value $\overline{R}$ of R_i, $i = 1, ..., N$
- estimating with Eq. 4 the number of exact decimal significant digits of $\overline{R}$.

In practice $\beta = 0.05$ and $N = 3$. Indeed, it has been shown [20], [21] 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 [5], [20].

B. Validity of the CESTAC method

Eq. 1 and 4 hold if the two following hypotheses are verified.

1) the round-off errors α_i are independent, centered uniformly distributed random variables,
2) the approximation to the first order in 2^{-p} is legitimate.

Concerning the first hypothesis, with the use of the random arithmetic, round-off errors α_i are random variables, however, in practice, they are not rigorously centered and in this case Student's test gives a biased estimation of the computed result. It has been proved [20] that, with a bias of a few σ, the error on the estimation of the number of exact significant digits of $\overline{R}$ is less than one decimal digit. Therefore even if the first hypothesis is not rigorously satisfied, the estimation obtained with Eq. 4 is still correct up to one digit.

Concerning the second hypothesis, the approximation to the first order only concerns multiplications and divisions. Indeed the round-off error generated by an addition or a subtraction does not contain any term of higher order. It has been shown [20] that, if both operands in a multiplication or the divisor in a division become insignificant, *i.e.* with no more exact significant digit, then the first order approximation may be not legitimate. In practice, the CESTAC method requires, during the execution of the code, a dynamical control of multiplications and divisions, which is a so-called *self-validation* of the method. Because insignificant operands in a multiplication may invalidate the estimation of accuracy, arguments of the power function must be controlled too. This self-validation leads to the synchronous implementation of the method, *i.e.* to the parallel computation of the N results R_i, and also to the concept of computational zero, also named informatical zero [22].

Definition 2.1: During the run of a code using the CESTAC method, an intermediate or a final result R is a computational zero, denoted by @.0, if $\forall i, R_i = 0$ or $C_{\overline{R}} \leq 0$.

Any computed result R is a computational zero if either $R = 0$, R being significant, or R is insignificant.

C. Principles of DSA

Discrete Stochastic Arithmetic (DSA) [6] has been defined from the synchronous implementation of the CESTAC method. With DSA, a real number becomes an N-dimensional set and any operation on these N-dimensional sets is performed element per element using the random rounding mode. The number of exact significant digits of such an N-dimensional set can be estimated from Eq. 4. From the concept of computational zero, an equality concept and order relations have been defined for DSA.

Definition 2.2: Let X and Y be N-samples provided by the CESTAC method.

- Discrete stochastic equality denoted by $ds=$ is defined as $X ds= Y$ if and only if $X - Y = @.0$.

- Discrete stochastic inequalities denoted by $ds>$ and $ds\geq$ are defined as:
  $Xds> Y$ if and only if $\overline{X} > \overline{Y}$ and $Xds\neq Y$,
  $Xds\geq Y$ if and only if $\overline{X} \geq \overline{Y}$ or $Xds= Y$.

Stochastic relational operators ensure that in a branching statement the same sequence of instructions is performed for all the samples which represent a variable. DSA enables to estimate the impact of rounding errors on any result of a scientific code and also to check that no anomaly occurred during the run, especially in branching statements.

III. The sequential version of the CADNA software

The CADNA software [17]–[19] 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 essence, classical floating-point variables are replaced by the corresponding stochastic variables, which are composed of three perturbed floating-point values. The library contains the definition of all arithmetic operations and order relations for the stochastic types. The control of the accuracy is performed only on variables of stochastic type. When a stochastic variable is printed, only its exact significant digits appear. For a computational zero, the string "@.0" is printed. Because all operators are overloaded, the use of CADNA in a program requires only a few modifications: essentially changes in the declarations of variables and in input/output statements.

CADNA can detect numerical instabilities which occur during the execution of the code. When a numerical anomaly 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; unstable power, where one of the arguments of the power function is a computational zero; and unstable division were 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 analyzed, 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 instabilites or to deactivate the detection of instabilities.

In the CADNA library, an integer variable named *accuracy* is associated with any stochastic number to store its number of exact significant digits. Thanks to this variable, the accuracy of a stochastic variable which is used several times without being modified is computed only once. This happens for instance in a matrix multiplication where the same stochastic variable acts as an operand in several multiplications that have to be controlled. The use of the *accuracy* variable increases the cost of CADNA in terms of memory, but reduces its cost in terms of execution time.

IV. Parallelization of DSA

A. Processes and data exchange

The basic idea is to run in parallel the three arithmetic operations inherent to any stochastic operation. The parallel architecture of CADNA is based on three UNIX processes, each one running an instance of the program. Processes exchange information through a communication system. Functions and operations that require data exchange between the processes can be classified in two groups.

1) The first group contains all functions which require the synchronization of the three processes. When executing these functions, processes exchange values synchronously to ensure they all compute the same result. Indeed the next sequence of instructions to be executed may depend on the result of such functions. This first group of functions includes:
   - equality and order relational operations
   - the absolute value function
   - conversion functions from a stochastic type to a classical floating-point type
   - functions which compute the number of exact significant digits of results (and therefore functions which display results with their accuracy).
2) The second group contains operations for which a part of the computation can be performed later. This group includes multiplications and divisions. As mentioned in Sect. II, the self-validation of the CESTAC method requires to check that both operands in multiplications and divisors are significant. Performing this kind of control before any multiplication or division would be costful, because it would lead to many communications. However this control can be postponed, because it has no impact on the choice of the next computation sequence to be executed.

The multicore versions of CADNA presented in this paper use UNIX functionalities. Two versions are described: in the first one, all data exchanges are performed synchronously; in the second one, only necessary synchronizations are performed. As a remark, once a program has been modified to be controlled using DSA, it can be executed with the sequential or the multicore versions of CADNA. The choice of the CADNA version is specified just for compiling and linking, it induces no extra change in the source code.

B. Version with synchronous data exchange

Three instances of a program are executed in parallel by three processes with the random rounding mode. When data exchange is required, it is performed synchronously. In this case, each process sends to the two others all values required by the function or the operation to be performed. Since all the processes own all the values, they all perform the same computation and take the same decision. Because associativity is not satisfied in IEEE floating-point arithmetic [23], the order of the three samples must absolutely be the same on the three processes. Otherwise processes may take different decisions and perform different sequences of instructions.

When the program starts, one process is launched. The CADNA initialization function creates a shared memory segment and uses the *fork* UNIX function to launch two other identical processes. Because the three processes must exchange values through the shared memory, attention has been paid to manage concurrent access to a unique memory zone. In order to optimize waiting time, a communication system with active waiting has been implemented, rather than UNIX semaphores.

Fig. 1 shows how a user program is executed. Each process created by the CADNA initialization function runs an instance of the same program. All tests require data exchange. Here, because the equality test is overloaded, discrete stochastic equality is actually tested in accordance with definition 2.2. Each process computes the difference between its operands. The three differences are exchanged through the shared memory segment, for the three processes to have exactly the same triplet $D = (D_1, D_2, D_3)$. The number $C_{\overline{D}}$ of exact significant digits of D is computed by all processes using Eq. 4. In all processes, the order of the values which compose the triplet must be the same for the same result $C_{\overline{D}}$ to be computed. This constraint is due to the fact that associativity is not necessarily satisfied in IEEE floating-point arithmetic and ensures that the result of the test and consequently the branch chosen are the same for the three processes.

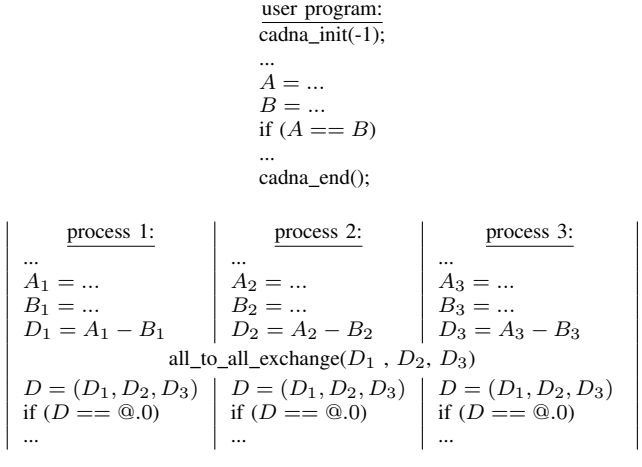

Fig. 1. Execution of a program using multicore DSA

C. Version using a validation box

In the second multicore version of CADNA, all data exchanges are not performed synchronously. Synchronizations are required for functions or operations in the first group, according to the classification introduced in IV-A. For operations in the second group (for instance multiplications and divisions) synchronizations are useless. Indeed the control of a multiplication or a division does not need to be performed just before the operation and can be postponed. In this version, computation inherent to the application and computation related to the self-validation of the CESTAC method are taken in charge by different processes. Like in the version described in IV-B, three processes execute three instances of the program. These processes form the so-called *computation box*. The self-validation, which consists in controlling the accuracy of operands in multiplications and divisions, is performed by one or several processes denoted by the *validation box*. For seek of simplicity, let us assume in the sequel that the validation box consists of one process. Processes in the computation box fill two buffers:

- a buffer to detect unstable multiplications which contains all multiplication operands
- a buffer to detect unstable divisions which contains all divisors.

The process in the validation box, which is created by a call to the *fork* UNIX function, reads the stochastic numbers in these buffers, checks their accuracy and, if necessary, increments the counter associated with unstable multiplications or the one associated with unstable divisions. If several processes form the validation box, they manage several buffers of both types. The first process, which is created using the *fork* UNIX function, lanches of a number of OpenMP threads specified by the user.

V. Performance analysis

A. Experimental environment

Because the parallel implementations of DSA examined in this section require three or four processes, experiments have been carried out on a quad-core processor. All execution times have been measured on a quad-core Intel i5-2500 processor running Linux with the gcc 4.6.3 compiler. Several implementations of DSA have been compared in terms of performance:

- the sequential CADNA library
- the parallel implementation of DSA with synchronous data exchange, as described in IV-B
- the parallel implementation of DSA with a validation box consisting of one process, as described in IV-C
- the parallel implementation of DSA with a validation box consisting of one process and an *accuracy* variable associated with any stochastic number; in the two other parallel implementations, the accuracy of a stochastic number may be computed several times even if this number is not modified.

All these implementations have been tested using several modes for the detection of numerical instabilities:

- the detection of all kinds of instabilities, denoted by the *full* mode;
- the detection of unstable multiplications and unstable divisions in order to perform a self-validation of the CESTAC method, as described in II;
- 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 cancellations is very costful in the sequential CADNA library. Therefore, in order to save communication time, it has been chosen not to enable this detection in the parallel implementations of DSA. For a complete numerical debugging of a code, the sequential CADNA library is preferable.

We are interested in the ratio between the execution time measured using the classical IEEE floating-point arithmetic and using DSA. The ratios obtained with the sequential CADNA library and several parallel implementations of DSA are compared for linear algebra programs in V-B and the computation of integrals in V-C.

B. Linear algebra programs

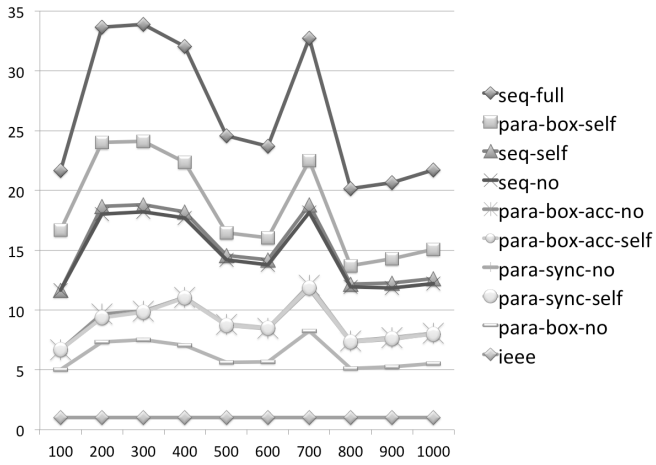

Fig. 2. Ratios between the execution times in IEEE arithmetic and in DSA for matrix multiplication, the matrix sizes varying from 100 to 1000.

Fig. 2 (respectively Fig. 3) presents the ratios between the execution times in IEEE arithmetic and in DSA for matrix multiplication (respectively for linear system solving using Jacobi method), the matrix sizes varying from 100 to 1000. In both figures, the graphs in the legend are ordered like in the chart. Some graphs are associated with very close execution times and therefore overlap.

From Fig. 2, for matrix multiplication, the cost of the sequential CADNA library varies from 12 to 19 with self-validation and from 20 to 34 with the detection of all kinds of instabilities. Matrix multiplication is implemented using three nested loops. Better performance would be obtained with an optimized routine in floating-point arithmetic and consequently the cost of CADNA would be higher.

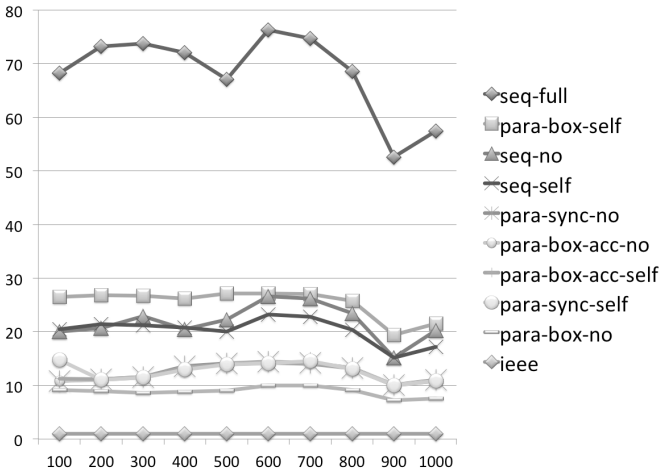

Fig. 3. Ratios between the execution times in IEEE arithmetic and in DSA for linear system solving using Jacobi method, the matrix sizes varying from 100 to 1000.

At each iteration of the Jacobi method, the infinite norm of the residue is computed. However since we are interested in performance comparison, the number of iterations is set to 328, which is, in this numerical experiment, the optimal number [22] determined thanks to the CADNA library. From Fig. 3, for Jacobi method, the cost of the sequential CADNA library varies from 15 to 23 with self-validation and from 53 to 76 with the detection of all kinds of instabilities.

The following remarks are valid for both programs. The execution times of the sequential CADNA library without instability detection and with self-validation are similar. This is partly due to the fact that the input matrices contain no computational zero. When the detection of all instabilities is activated, the increase of the execution time is essentially due to the detection of cancellations.

With the parallel implementation of DSA with synchronous data exchange, the execution times are similar whatever the mode chosen for instability detection. We recall that the detection of cancellations is not implemented in this parallel version. The cost of this parallel implementation of DSA is reduced by a factor of about 2, compared with the sequential CADNA library with self-validation.

The parallel implementation with the validation box and no *accuracy* variable performs worse than the sequential CADNA library, when self-validation is activated. As a remark, the performance of this parallel implementation without any instability detection is very satisfactory. However the self-validation is strongly recommended, otherwise results may be displayed with a too optimistic accuracy without any warning message.

The parallel implementation with the validation box and the *accuracy* variable are similar to those of parallel implementation of DSA with synchronous data exchange.

C. Computation of integrals

Tables I and II present the execution times of programs computing integrals using the trapezoidal method. A sequence of

Execution	instability detection	execution time (s)	ratio
IEEE	-	8.80	1
sequential DSA	full	94.00	10.7
	self-validation	66.17	7.5
	no detection	57.57	6.5
parallel DSA (synchronous exchange)	full	84.03	9.5
	self-validation	56.73	6.4
	no detection	30.59	3.5
parallel DSA (validation box)	full	59.92	6.8
	self-validation	35.11	4.0
	no detection	28.06	3.2
parallel DSA (validation box & accuracy)	full	59.72	6.8
	self-validation	32.28	3.7
	no detection	32.24	3.7

TABLE I
EXECUTION TIME FOR THE COMPUTATION OF I_1

Execution	instability detection	execution time (s)	ratio
IEEE	-	0.22	1
sequential DSA	full	40.18	182.6
	self-validation	28.15	128.0
	no detection	20.02	91.0
parallel DSA (synchronous exchange)	full	17.81	81.0
	self-validation	17.91	81.4
	no detection	10.96	49.8
parallel DSA (validation box)	full	23.10	105.0
	self-validation	23.09	105.0
	no detection	8.71	39.6
parallel DSA (validation box & accuracy)	full	10.88	49.5
	self-validation	10.85	49.3
	no detection	10.81	49.1

TABLE II
EXECUTION TIME FOR THE COMPUTATION OF I_2

approximations is computed, the integration step being halved at each iteration. The execution times have been measured for 2^{24} partitions of the integration interval. We are interested in performance comparisons; readers can refer to [7], [24]–[26] for detailed information about the dynamical control of integrals computation using DSA and strategies for optimizing the global error which consists of both the truncation error and the round-off error. The following integrals have been computed.

- $I_1 = \int_1^{100} f_1(x)dx$
  with $f_1(x) = \frac{\sin(x)}{x} + \cos(x)\exp(\sin(x))$
- $I_2 = \int_{-1}^{2} f_2(x)dx$
  with $f_2(x) = \frac{2x^5-10x^4+5x^3-60x^2+80x+37}{8x^4+13x^3-38x^2+43x+513}$

The cost of DSA depends strongly on the integrand. The ratio between the execution times measured using the classical IEEE floating-point arithmetic and using the sequential version of CADNA with self-validation is 7 for I_1 and 128 for I_2. The integrand f_2 is particularly unfavourable to DSA, because it contains mathematical expressions that are efficiently computed in one instruction using IEEE floating-point arithmetic.

The following remarks are valid for both integrals. The performance obtained using a parallel implementation of DSA with a validation box, no *accuracy* variable and no instability detection is very satisfactory. However at least self-validation is strongly recommended. For each implementation of DSA, execution times measured with no instability detection should be considered the minimum that can be obtained. If self-validation is activated using a parallel implementation of DSA, the best performance is obtained with a validation box and an *accuracy* variable. Using this configuration, the cost of DSA with self-validation is reduced by a factor of at least 2 compared with the sequential CADNA library, whatever the integral.

VI. NUMERICAL VALIDATION OF THE SHALLOW-WATER APPLICATION

The sequential CADNA library and the parallel implementations of DSA described in Sect. IV have been integrated into the shallow-water application [27] developed by the LOCEAN laboratory in Paris for data assimilation experiments. This application implements the two-dimensional shallow-water model in the horizontal plane (x,y), also called Saint-Venant model, which arises from the vertical integration of three-dimensional Navier-Stokes equations. This model describes the linear flow of a nonviscous fluid in shallow-water environment with a free surface. The shallow-water application contains over 8,000 lines of codes.

212 unstable multiplications and 149,564 losses of accuracy due to cancellations have been detected by the CADNA library. All the operands responsible for the unstable multiplications have been identified. Their order of magnitude is such that they have no impact on the estimation of accuracy provided by CADNA. The cancellations have a limited impact on the numerical stability of the code, which is very satisfactory.

Execution	instability detection	execution time (s)	ratio
IEEE	-	7.76	1
sequential DSA	full	192.38	24.8
	self-validation	70.64	9.1
	no detection	70.65	9.1
parallel DSA (synchronous exchange)	self-validation	41.34	5.3
	no detection	19.42	2.5
parallel DSA (validation box)	self-validation	25.28	3.3
	no detection	16.75	2.2
parallel DSA (validation box & accuracy)	self-validation	20.17	2.6
	no detection	20.19	2.6

TABLE III
EXECUTION TIME OF THE SHALLOW-WATER APPLICATION

From Table III, the cost of the sequential CADNA library is a factor of about 25 if all kinds of instabilities are detected. It is a factor of about 9 with self-validation only. With the parallel implementations of DSA, if self-validation is activated, the cost of the numerical validation is reduced by a factor of 1.7 to 3.5, the best performance being measured with the validation box and the *accuracy* variable. As a remark, in the parallel implementations of DSA, because the detection of cancellations is not enabled, the execution times with the detection of all kinds of instabilities and with self-validation are similar. The moderate cost of DSA is due to the fact that

the shallow-water application performs not only computation but also input/output tasks. As observed in Sect. V, the cost of DSA on a computation kernel may be high. However this cost usually becomes reasonable on a real-life application.

VII. Conclusion

In this paper, we have shown that we can take advantage of multicore architectures to improve the performance of DSA. Several parallel implementations of DSA which use UNIX functions for data exchange have been compared. The best performance has been obtained with the following features: a validation box for the detection of instabilities and a variable to store the accuracy of results. With such a parallel implementation, the cost of DSA is reduced by a factor of about 2 compared with the sequential CADNA library when the same level of instability detection is enabled. With the shallow-water application, we have shown that using our parallel implementation the cost of DSA can be significantly reduced on real-life applications.

In our parallel implementation of DSA, the detection of cancellations is not possible because it would require too many data transfers. The same modifications in a program are required by the sequential CADNA library and our parallel implementation of DSA. Therefore we recommend the following strategy for the numerical validation of a sequential program. First it can be executed with our parallel implementation of DSA to check the numerical quality of its results. Then, for a more detailed analysis, it can be executed with the CADNA library. In this case, the instructions responsible for numerical instabilities can be identified using a debugger and, if possible, modified to improve the numerical quality of the results.

A version of CADNA for the numerical validation of MPI programs already exists. The present paper shows it will be possible to improve its performance when it is used on multicore processors. The numerical validation of OpenMP programs, which is still under investigation, may also benefit from the comparison of the different strategies reported in the present paper.

Acknowledgement

The authors wish to thank J. Brajard from the LOCEAN laboratory in UPMC (Université Pierre et Marie Curie, Paris, France) who made the shallow water application available and P. Li who controlled with CADNA the shallow water application during his internship in the framework of the Emergence-UPMC research program.

References

[1] J. Wilkinson, *Rounding errors in algebraic processes*. Englewood Cliffs, N.J.: Prentice-Hall Inc., 1963.

[2] N. Higham, *Accuracy and stability of numerical algorithms*, 2nd ed. Philadelphia, PA: Society for Industrial and Applied Mathematics (SIAM), 2002.

[3] G. Alefeld and J. Herzberger, *Introduction to interval analysis*. Academic Press, 1983.

[4] U. Kulisch, *Advanced Arithmetic for the Digital Computer*. Springer-Verlag, Wien, 2002.

[5] J. Vignes, "A stochastic arithmetic for reliable scientific computation," *Math. Comput. Simulation*, vol. 35, pp. 233–261, 1993.

[6] ——, "Discrete stochastic arithmetic for validating results of numerical software," *Num. Algo.*, vol. 37, no. 1–4, pp. 377–390, Dec. 2004.

[7] 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.

[8] 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.

[9] J.-L. Lamotte, "Vers une chaîne de validation des logiciels numériques à l'aide de méthodes probabilistes," Habilitation à diriger des recherches, Université Pierre et Marie Curie, Paris, Nov. 2004.

[10] S. Montan and C. Denis, "Numerical Verification of Industrial Numerical Codes," in *ESAIM: Proc.*, vol. 35, Mar. 2012, pp. 107–113.

[11] F. Jézéquel and J.-L. Lamotte, "Numerical validation of Slater integrals computation on GPU," in *14th international symposium on Scientific Computing, Computer Arithmetic and Validated Numerics (SCAN 2010)*, Lyon, France, Sep. 2010, pp. 78–79.

[12] 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)*, july 2011, pp. 510–516.

[13] J.-L. Lamotte, "Parallelization of the CESTAC method on shared memory and distributed memory computers," in *10th International Symposium on Scientific Computing, Computer Arithmetic, and Validated Numerics (SCAN 2002)*, Paris, France, Sep. 2002.

[14] A. Ghaneme and J.-L. Lamotte, "On the performance of a parallel implementation of the CESTAC method with self-validation on several parallel machines," in *Proc. SCAN2004 conference, Fukuoka, Japan*, Oct. 2004.

[15] J.-L. Lamotte and D. Martins, "First parallel implementation of the CESTAC method with self validation," Mathematical Modelling and Scientific Computations, minisymposium within the 33rd Spring Conference of the Union of the Mathematicians in Bulgaria, Borovets, Bulgaria, pp. 427–433, Apr. 2004.

[16] W. Li and S. Simon, "Numerical error analysis for statistical software on multi-core systems," in *Proceedings of COMPSTATS 2010, 19th International Conference on Computational Statistics, Paris - France, August 22-27*, Y. Lechevallier and G. Saporta, Eds. Physica-Verlag HD, 2011, pp. 1295–1302.

[17] 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.

[18] 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.

[19] 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.

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

[21] J.-M. Chesneaux and J. Vignes, "Sur la robustesse de la méthode CESTAC," *C. R. Acad. Sci. Paris Sér. I Math.*, vol. 307, pp. 855–860, 1988.

[22] J. Vignes, "Zéro mathématique et zéro informatique," *C. R. Acad. Sci. Paris Sér. I Math.*, vol. 303, pp. 997–1000, 1986, also: La Vie des Sciences, 4 (1) 1-13, 1987.

[23] *IEEE Standard for Binary Floating-Point Arithmetic, ANSI/IEEE Standard 754-1985*, IEEE Computer Society, New York, 1985, reprinted in SIGPLAN Notices, 22(2):9-25, 1987.

[24] F. Jézéquel, "A dynamical strategy for approximation methods," *C. R. Acad. Sci. Paris - Mécanique*, vol. 334, pp. 362–367, 2006.

[25] F. Jézéquel and J.-M. Chesneaux, "Computation of an infinite integral using Romberg's method," *Num. Algo.*, vol. 36, no. 3, pp. 265–283, Jul. 2004.

[26] F. Jézéquel, "Dynamical control of converging sequences computation," *Applied Numerical Mathematics*, vol. 50, no. 2, pp. 147–164, 2004.

[27] L. Nardi, C. Sorror, F. Badran, and S. Thiria, "YAO: A Software for Variational Data Assimilation Using Numerical Models." in *LNCS 5593, Computational Science and Its Applications - ICCSA 2009*, O. Gervasi, D. Taniar, B. Murgante, A. Laganà, Y. Mun, and M. L. Gavrilova, Eds. Springer-Verlag, 2009, pp. 621–636.