Reproducibility of simulations and High Performance Computing

David R.C. Hill

REPRODUCIBILITY OF SIMULATIONS AND HIGH PERFORMANCE COMPUTING

David R.C. Hill
Clermont-Auvergne-INP, CNRS, Mines de Saint-Étienne
Université Clermont-Auvergne
LIMOS UMR 6158 – ISIMA – F-63000 Clermont-Ferrand, FRANCE.
E-mail: David.Hill@uca.fr

KEYWORDS
Reproducibility, Repeatability, Replicability, Simulation, High Performance Computing.

ABSTRACT

One of the major criteria for the scientificity of a research study is reproducibility. In this paper, we will present the main definitions around reproducibility. We will examine to what extent computer science and simulation works are concerned. Anyone wishing to produce quality scientific work will pay attention to the numerical reproducibility of his simulation results. Significant differences can be observed in the results if the practitioner does not apply best practices. In addition to reproducibility, we first need repeatability to debug our simulation software. We will see that even for parallel stochastic simulations, it is possible to reproduce the same numerical results by implementing a rigorous method tested with a billion threads. It is possible to check parallel results with their sequential counterpart before scaling, thus gaining confidence in the proposed simulation results.

INTRODUCTION

Karl Popper is a major reference in epistemology. He had a profound impact on the production of knowledge in many fields (Popper 2005). Recent drifts in scientific practices have led society to realize the importance of the criteria of scientificity that Popper had put forward for experimental Science. One of the major criteria is the reproducibility of scientific experiments. In 2005, John Ioannidis, of Stanford University, suggested provocatively in an article in PloS Medicine that “most scientific results are false” and he explained that it is because many of them are impossible to reproduce (Ioannidis 2005).

Ten years after, the Guardian reported on the work financed in Great Britain by the fund for scientific excellence. Even there, the reproducibility rate for scientific work in the medical field was extremely weak (around 11%). A recent manifesto in one of the Nature journal series declared that reproducing scientific results is difficult and time-consuming, but it is essential for the advancement of Science, thus the effort has to be shared between the authors, the laboratories and the scientific community (Munafò et al. 2017).

In Computer Science we could expect much better "performances" because our machines and our software stacks are always supposed to be deterministic. However, an extensive study, dating from the same period, showed that the reproducibility of our research works in Computer Science hardly exceeded 30% (Collberg and Proebsting 2016) (Bisgambiglia and Hill 2022). Indeed, obtaining the results presented in computer-based research study is often problematic due to limited documentation. Moreover, the main reason why we cannot reproduce the results of a computer based research study is clearly the lack of access to the source code. Computer simulations have developed to the point of becoming a valuable if not an essential tool for the production of knowledge and decision support. For complex systems, it is the only tool available to explore what our mind could not imagine. In this context, even elementary deterministic models surprisingly and flagrantly show the limits of our knowledge (Zwirn 2000; 2006). This emphasizes the importance of simulations to explore the functioning of these so-called complex systems (Wolfram 2018). For instance, Christopher Langton proposed with his “ant” a two-dimensional cellular automaton with an excessively simple set of rules whose surprising behavior in 3 phases can only be observed through a simulation: symmetrical, chaotic and then infinitely drifting (Gajardo et al. 2002). Though many decision processes rely on computer simulation results, their progress in terms of ergonomics and ease of use leads to the fact that they can be used without enough hindsight on the underlying mechanisms: be it the resolution or discretization methods, the pseudo-random number generators used, event schedulers, concurrency management mechanisms, compilation mechanisms, etc. This is mainly because the software stack that allows an application to be used is assumed to be mastered. But this is not always the case for too many sophisticated scientific applications. Nowadays, in many modern software chains or workflows, it is no longer "obvious" to obtain two identical software during two successive builds (Lamb and Zacchiroli 2021).

We will first present the main definitions encountered in reproducible research. This field is quite new for computer scientists since reproducibility was considered granted. In the past twenty years, we have seen an evolution in the terminology which is sometimes confusing. Then, we will address the reasons why we could fail to reproduce our simulation results in the context of High Performance Computing. Eventually, we will present a technique enabling the production of comparable results even with parallel stochastic simulations. When stochastic simulations are used, this technique is able to face the ‘silent errors’ impacting top supercomputers.

REPRODUCIBLE RESEARCH AND COMPUTER SCIENCE

First, we want to define the notion of computer repeatability. It means that we want bitwise identical results. If you don't have this property, how do you debug?

It is sometimes called bitwise reproducibility which adds confusion, we will discuss the terminology evolution. As said above, this property is essential for debugging, but it has only a little to do with reproducibility in the epistemological sense used by Popper. If we lose run-to-run repeatability on the same machines, our computers are worthless – they lose their key deterministic property. This case is very rare for common applications, but unfortunately in the case of High Performance Computing (HPC) this is occurring more often than we would like to, and we have to "fight" to come back to a reliable hardware/software couple enabling the restoration of confidence in what the simulation outputs.

Reproducing a result in the epistemological sense means that a different laboratory, starting the experiment from scratch, would get the same scientific conclusion (and not exactly the same results). It means you have a corroboration which increases your confidence in the work achieved by some independent pairs. Your scientific results and the scientific conclusion obtained by other authors statistically match. The important thing is to have an independent team following the same protocol. But you can obtain even more confidence when other things have changed in the way the study has been conducted.Another machine, another computer language, another software stack or method enabling the finding of the same scientific conclusion for the same problem. This is augmenting the confidence we have in a scientific finding and it will lead to a new set of definitions as presented later.

An important pioneering work was carried out at Stanford by Claerbout and Karrenbach in 1992, an update can be found in (Schwab et al. 2000) to discover the initial definitions. More recently, the work of Stodden at MIT (Stodden 2009) and our work (Hill 2015) discussed definitions in the context of scientific computing. The terms "reproduce" and "replicate" often meant different things and sometimes were found interchangeable even with the same authors. A paper by Drummond in 2009 in the machine learning field added confusion. At the same time, following Claerbout, Donoho and Stodden (Donoho et al. 2009), reproducible research in Computer Science linked the production of research papers to the data and all the software tools allowing a reader to find the results that are presented. All what is needed in addition to the research paper is called the research artifact : the entire software system or workflow used to run the digital experiments and produce results, the input datasets, the output raw data collected during the experiments and the software used for analysis and needed to obtain the final results. Even though this is very interesting and essential, this is limiting the epistemological meaning of reproducibility where many things can change to corroborate the scientific results, including input data, methods and tools,…

Following this, a recent change in terminology was made by the ACM (Association for Computing Machinery) thanks to discussions with NISO (National Information Standards Organization). NISO recommended to ACM a harmonization with what is found not only in Computer Science but in the whole Scientific Community. This introduced a recent swap in the terms "reproducibility" and "replicability" to meet their recent and current version of Artifact Review and Badging. The reference (ACM 2020) gives the URL to find the full and current definitions. They are inspired by the International Vocabulary of Metrology since the result of an experiment can be seen as a measurement on a virtual entity (VIM 2012). All this evolution in terminology has been carefully reviewed by Lorena Barba in a paper dedicated to terminology (Barba 2021). We cannot establish experimental results if they can't be reproduced. The new definitions proposed by ACM in 2020 distinguish again the three terms, can of course still be discussed since the concern for reproducible research is rather new for computer experiments :

Repeatability

For the current ACM definition, repeatability implies that a computer scientist can repeat calculation and should find each time the same result with a stated precision (the same team and the same experimental setup).

Reproducibility

The latest ACM definition for reproducibility, says that a person or a group of researchers independent from the initial author (or group of authors) is able to obtain, with a stated precision, the same result with thanks to the initial author's artifact (a different team trying to obtain the same results using the same experimental setup).

Replicability

This term replicability is considered as rather new for many computer scientists (and dictionaries…). It means that a new team should obtain the same result, with a stated precision, using artifacts which they develop completely independently (a different team working with a different experimental setup).

The notion of "same results" remains vague. With a stated precision it fits with the requirements for measurements which inspired ACM, but it fails to meet the debugging requirements, essential for software development. This last point requires bitwise identical results from run-to-run when dealing with repeatability. Digital computers are built to be deterministic.

WHY DO WE LOSE REPEATABILITY AND REPRODUCIBILITY ?

We have already mentioned that the main reason why there is a lack of reproducibility in computer science is due to the fact that authors rarely share their code (and data). The

second important reason comes from our limit to implement real numbers and mathematical operations with bit registers. One can see the famous paper from Goldberg : “What every computer scientist should know about floating-point arithmetic” to understand the IEEE 754 standard (and its new updates in 2008 and 2019). This standard represents real numbers with fractions and comes with fast hardware implementations. However, the accumulation of rounding errors can impact some complex floating-point computing, particularly in high performance computing. Compensated summation algorithms like the one proposed by Kahan and his colleagues significatively reduces such errors and most of them can be parallelized. Recent works in this field were proposed by (Blanchard et al. 2020) and (Lange 2022).

Still with floating points, it is also sometimes, if not often, forgotten that the order in which floating point additions and multiplications are executed matters. While mathematically such operations are associative in the set of real numbers, it is not the case in the space of fractions. Optimization of the Intel C++ Compiler uses associativity-based transformations by default and this can impact your result (particularly with compensated sums). It is important to control the compiler options to be sure that the order of operations will follow the source code. For instance we can find advices in the Intel compiler documentation to improve numerical reproducibility. Indeed, many software or hardware optimizations could influence the final result. This is particularly true with the dynamic execution mechanism included in modern processors. The reordering proposed by such optimization could lead to repeatability problems from run-to-run when using floating point operations. Reproducibility and repeatability problems are often:

- related to the implementation in different programming languages,
- related to different compilers or compiler options even when the same programming language is used,
- linked to environment : different operating systems or different virtual machines or containers, different versions of compilers, compiler options, libraries,…
- due to a limited knowledge of the pseudorandom number used (poor or dated generators which do not meet 21st century computing requirements), bad or poor initialization of the pseudorandom number generator status, bad or inadapted parallelization technique used for the pseudorandom number generator.
- related to hardware diversity : GP-GPUs, FPGAs, MPPA, Manycore of the Intel Xeon Phi type with a k1OM architecture instead of a regular x86 architecture (even when using the same compiler, the same language and the same operating system),
- observed from one execution to another on the same microprocessor while all the rest of the computer context is strictly identical! This is the worst case scenario, where we lose run-to-run reproducibility and unfortunately it is being observed more and more frequently due to dynamic execution on modern CPU architectures.

In the next section, we will give some good practice which enables the mastering of parallel stochastic simulations. With a rigorous method it is possible to keep the expected reproducibility, the statistical quality of the results by avoiding correlations between the parallel computations.

THE DESIGN OF REPRODUCIBLE PARALLEL STOCHASTIC SIMULATIONS

The “deterministic” nature of “stochastic” simulations

We sometimes read in high performance computing magazines that it is normal to obtain different results for the same parallel stochastic simulations because of stochasticity. It is important to remember that pseudorandom numbers, though they simulate randomness, are deterministic programs. They are precisely designed to be repeatable in order to be able to debug stochastic programs (Monte Carlo simulations and all their derivatives). In the context of so-called “in order” processors (which could become rare), the hardware at our disposal does not dynamically change the order of instructions and can be considered deterministic. Development environments and compilers are also designed to be still deterministic unless we explicitly ask the compiler to try "unsafe" optimizations. Thus we should be able to obtain identical results from one stochastic simulation to the other when the input data are the same. Not being able to reproduce its results from one execution to another highlights either a hardware or software problem, or a lack of rigor in the scientific design of the study.

A short list of modern and parallelizable high quality generators with ‘independent’ streams

The algorithms that we have selected below all offer a reliable technique according to the best statistical test batteries for distributing random numbers in a parallel environment : TestU01 (L'Ecuyer and Simard 2007). We have already identified and detailed these techniques in (Hill et al. 2013), but only some of them can be applied on a GPU platform. Here is a selection of some interesting generators:

- Mersenne Twister (Matsumoto and Nishimura 1998) is a generator which quickly became a reference. Despite some known defaults (don’t use it for cryptography), it is fast for simulations (faster version exists with SIMD instructions). Its huge period 2^{19937} makes it interesting for partitioning with various techniques for exascale machines cores and for experimental designs in high dimensions.
- MRG32k3a (L’Ecuyer 1999). This concise algorithm was intended to be implemented directly on floating-point real numbers. Its data structure only stores 6 values in double precision. It is possible to divide the initial sequence into 2^{64} adjacent streams each containing 2^{127} elements. Its optimized implementation on modern Intel C/C++ type compilers is up to 15 times slower than Mersenne Twister with recent Intel Xeons and an Intel compiler. MRG32K3a remains suitable for HPC when the proportion of calculation associated with the volume of pseudo-random number drawings is low.
- MLFG6331_64 (Mascagni and Srinivasan 2004). It is a 64-bit nonlinear Multiplicative Lagged Fibonacci Generator (MLFG) type generator. With a period of 2^{124},

it can provide, like MRG32k3a, more than 2^{60} parallel streams thanks to a parameterization mechanism.

- Phylox and Threefry (Salmon et al. 2011). Salmon and his colleagues proposed three cryptographically inspired generators for CPU and GPU at the 2011 supercomputing conference. They also rely on the technique of parametrization to solve the problem of distribution of stochastic fluxes within parallel applications.

For parallel stochastic simulations we need indenpendent streams to carry out (simulate) 'independent' experiments. There will never be any mathematical proof of this 'independence', the proof of the dependence being the source code of the generator itself. However, the best mathematicians have worked on techniques that allow us to assign different 'independent' random streams for different elements of parallel computing with one of the following two approaches (1) partitioning of a single random stream and (2) producing multiple independent streams. In the latter case, the independent fluxes are obtained by parameterization techniques designed depending on the "family" of generators. Much more details dealing with distribution and parallelization techniques can be found in (Hill et al. 2013).

A METHOD FOR REPEATABLE PARALLEL STOCHASTIC SIMULATIONS

In (Hill 2015) we proposed a method allowing the repeatability of the results from parallel stochastic simulations. With this method, it is possible to compare the results of the parallel program with a sequential version of the program which serves as a reference (at a smaller scale). To obtain repeatable results, we give the following recommendations to designers:

- use an object-oriented simulation approach (Hill 1996);
- use a high quality generator such as those mentioned above;
- use of a fine parallelization technique for retained pseudorandom number generator;
- use a "parallel" design for the sequential program which serves as a reference.

This last part is key to gain confidence in the parallel version of the simulation. It has to be tested at small scales, for instance at the scale of a node in a parallel machine. The details of the method is given hereafter:

1. First, design a sequential program with its input data set that will serve as a reference. The key is based on the fact that this design of the sequential program must be done by thinking parallel from the start. All the "stochastic objects" of this sequential simulation can be designed to be executed independently and therefore when going parallel, the result will not be impacted. Each stochastic object identified has its own stream of pseudorandom numbers with a precise initial status to replay the simulation.
2. The set of "independent" stochastic streams must be produced using one of the techniques previously discussed in (Hill et al. 2013). As specified above, all the initial statuses of the stochastic flows used should be archived in the initial state of the simulation, so as to be able to reproduce the same sequence for each simulated stochastic object.
3. The parallel execution will be based on a mapping of the independent stochastic objects over a number N of computing units (or PE, Processing Elements). Whatever the number of PEs, the stochastic flows are specific to each object and will be initialized with their own status, the same statuses used in the sequential simulation.
4. At a small scale (typically the scale of a node of a parallel machine), check that parallel executions on different numbers of Processing Elements give the same primary results as the reference sequential execution when the same input dataset is used.
5. When the previous stage of the method is achieved. The simulation can be mapped to many nodes.
6. When intermediate results are obtained in parallel, the reduction phase can use compensated sums when needed. The computing of the final reduction should respect the same order of operations which was used by the sequential program (the Processing Element ID can help to keep the same order each time we compute the reduction). Thus the calculation of an average and its confidence interval can lead to the same floating point result..

Using this method ensures that the results of stochastic parallel simulations are repeatable, enabling thorough debugging. This property is very important for many scientists working in sensitive fields: medicine, nuclear safety, finance, etc. We applied this method successfully since 2015 within the framework of the LabEx Clervolc and the Tomuvol project : Muonic simulations for Computed Tomography of volcanoes. When using large batches of parallel independent stochastic simulations on huge top machines, silent or 'soft' errors (alpha particles for instance) are impacting some computing nodes. They can be detected statistically and removing as 'bad' elementary results (measures) to save the entire computing; a full deterministic code would have failed.

CONCLUSION

This article aims at increasing the sensitivity of our scientific community to the need for reproducibility. It aims to increase the quality of our publications and save the time spent trying to reproduce the results of studies that are not sufficiently documented. We have seen the evolution in the terminology of this field in the past twenty years and we have given a focus on High Performance Computing (HPC) with the case of stochastic parallel simulations. Monte Carlo simulations are very common, particularly in Physics where we have proposed reproducible approaches, with a year of data collected on the Worldwide LHC Computing Grid (Boyer et al 2022a). We have proposed a method which enables the repeatability and reproducibility of scientific results and in addition we are able to compare the sequential and parallel results. We will end this conclusion with a warning related to the use of hybrid machines using different computing hardware architectures (in particular calculation

accelerators). We addressed this case with classic Xeon processors and the initial manycores (Xeon Phi) still in use on some supercomputers (Boyer et al. 2022b). Without cautions, the differences in the results can reach several orders of magnitude. With regard to GP-GPUs, Taufer notes that these architectures do not easily allow obtaining satisfactory numerical reproducibility (Taufer et al. 2010). There is still a lot to do to achieve reproducible research with hybrid computing.

ACKNOWLEDGMENTS

We thank Benjamin Antunes M.Sc. and Dr. Alexandre Boyer for their assistance with proof-reading.

REFERENCES

Barba L.A. 2018. “Terminologies for reproducible research”. arXiv preprint arXiv:1802.03311.

Bisgambiglia P.A. and Hill D., 2022, "Reproductibilité numérique : enjeux de crédibilité pour les expériences de simulation", BULLETIN de la société informatique de France 1024. No.19, 137-144.

Blanchard P.; N.J. Higham and T. Mary. 2020. “A class of fast and accurate summation algorithms”. In *SIAM Journal on Scientific Computing*. Vol. 42 No.3., A1541–A1557.

Boyer A., C. Haen; F. Stagni and D.R.C. Hill. 2022. “DIRAC Site Director: Improving Pilot-Job provisioning on grid resources”, In *Future Generation Computer Systems*, Vol. 133, 23-38.

Boyer A.; C. Haen; F. Stagni and D.R.C. Hill. 2022. “A Subset of the CERN Virtual Machine File System: Fast Delivering of Complex Software Stacks for Supercomputing Resources”, In *International Supercomputing Conference. ISC High Performance*, LNCS, Vol. 13289, 354–371.

Collberg C. and Proebsting T.A. 2016. “Repeatability in computer systems research”. *Communications of the ACM*, Vol. 59, No.3, 62-69.

Donoho D.L.; A. Maleki; I.U. Rahman; M. Shahram M. and Stodden V. 2009. “Reproducible research in computational harmonic analysis”. In *Computing in Science & Engineering*. Vol. 11 No.1, 8–18.

Goldberg D. 1991. “What every computer scientist should know about floating-point arithmetic”. In *ACM computing surveys*. Vol. 23, No.1, 5-48.

Hill D.R.C. 1996. *Object-Oriented Analysis and Simulation.* Addison-Wesley Longman.

Hill D.R.C.; J. Passerat-Palmbach; C. Mazel C. and M.K. Traore. 2013. “Distribution of random streams for simulation practitioners”. In *Concurrency and Computation: Practice and experience*. Vol. 25, No.10, 1427-1442.

Hill D.R.C. 2015. “Parallel Random Numbers, Simulation, and Reproducible Research”. In IEEE *Computing in Science and Engineering*. Vol. 17, No.4, pp. 66-71.

Ioannidis, J.P.A. 2005. “Why most published research findings are false”. *PLoS medicine*, Vol. 2, No.8 : e124.

Lamb C. and Zacchiroli S. 2021. “Reproducible builds : Increasing the integrity of software supply chains”. In *IEEE Software*. Vol. 39, No.2, 62-70.

Lange M. 2022. “Toward accurate and fast summation”. In *ACM Transactions on Mathematical Software*. Vol. 48 No.3, 1–39.

L’Ecuyer P. 1999. “Good parameters and implementations for combined multiple recursive random number generators”. In *Operations Research*, Vol. 47, No.1, 159-164.

L’Ecuyer P. and R. Simard R. 2007. “TestU01: A C Library for Empirical Testing of Random Number Generators”. In *ACM Transactions on Mathematical Software.* Vol. 33, No.4., 40 p.

Mascagni M. and A. Srinivasan A. 2004. “Parameterizing Parallel Multiplicative Lagged-Fibonacci Generators”, In *Parallel Computing*. Vol. 30, 899-916.

Matsumoto M. and T. Nishimura T. 1998. “Mersenne Twister: A 623-dimensionnally equidistributed uniform pseudorandom number generator”, In *ACM Transactions on Modeling and Computer Simulations: Special Issue on Uniform Random Number Generation*. Vol 8 No.1., 3-30.

Munafò M.R.; B.A. Nosek; D.V.M. Bishop; K.S. Button; C.D. Chambers; N. Percie du Sert; U. Simonsohn, E-J. Wagenmakers; J.J. Ware and J.P.A. Ioannidis. 2017. “A manifesto for reproducible science”. *Nature human behaviour*, Vol. 1, No1, 1-9.

Popper K. 2005. *The logic of scientific discovery*. Routledge, Basic Books Inc. New York.

Salmon J.K.; M.A. Moraes; R.P. Dror and D.E. Shaw. 2011. “Parallel random numbers: as easy as 1, 2, 3.”; In *Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis*, 1-12

Schwab M., Karrenbach M. et Claerbout J. 2000. “Making Scientific Computations Reproducible”. In *Computing in Science & Engineering*. Vol. 2, No.6., 61–67.

Stodden V. 2009. “The reproducible research standard: Reducing legal barriers to scientific knowledge and innovation”. In IEEE *Computing in Science & Engineering*. Vol. 11, No.1, 35-40.

Taufer M.; O. Pardon; P. Saponaro and S. Patel. 2010. “Improving Numerical Reproducibility and Stability in Large-Scale Numerical Simulations on GPUs”. In *IEEE International Symposium of Parallel and Distributed Processing*. 1–9

Wolfram S. 2018. “Complex Systems Theory”, In *Emerging syntheses in science*. CRC Press, 183-190.

Zwirn H. 2000. *Les limites de la connaissance*, Odile Jacob.

Zwirn H. 2006. *Les systèmes complexes*. Coll. Mathématiques et biologie, Odile Jacob.

WEB REFERENCES

ACM. 2020. Artifact Review and Badging Version 1.0 and link to V.1.1 (current - August 24, 2020) https://www.acm.org/publications/policies/artifact-review-badging (Accessed Oct. 5th 2022).

VIM. 2012. VIM: International Vocabulary of Metrology. https://jcgm.bipm.org/vim/en/index.html (Accessed Oct. 5th 2022).

BIOGRAPHY

DAVID R.C. HILL is doing his research at the French Centre for National Research (CNRS) in the LIMOS laboratory (UMR 6158). He earned his Ph.D. in 1993 and Research Director Habilitation in 2000 both from Blaise Pascal University and later became Vice President of this University (2008-2012). He is also past director of a French Regional Computing Center (CRRI) (2008-2010) and was appointed two times deputy director of the ISIMA Engineering Institute of Computer Science – part of Clermont Auvergne INP, #1 Technology Hub in Central France (2005-2007 ; 2018-2021). He is now Director of an international graduate track at Clermont Auvergne INP. Prof Hill has authored or co-authored more than 250 papers and has also published several scientific books. He recently supervised research at CERN in High Performance Computing.