Kullback-Leibler divergence as an estimate of reproducibility of numerical results

Florent Calvayrac

Kullback-Leibler divergence as an estimate of reproducibility of numerical results

Florent Calvayrac
Institut des Molécules et Matériaux du Mans
LUNAM UMR6283
Faculté des Sciences et Techniques Université du Maine
F-72085 Le Mans Cedex France
Email: Florent.Calvayrac@univ-lemans.fr

***Abstract*—In large software projects using numerical solutions of equations, small changes in compiler options or parallelization methods can induce slight variations in the last digits of floating point numerical results, due for instance to the non-commutativity of operations. Unfortunate changes in the code (bugs) can induce even larger deviations in the results. We propose to use the Kullback-Leibler divergence estimated from the compression ratio of the results compared to reference results in order to automatically quantify those changes and automatize regression tests of numerical codes in software forges. We use the TDDFT PW-TELEMAN project as an example.**

***Index Terms*—Kullback-Leibler, numerical, reproducibility, software forge**

I. Introduction

It is well known that *a priori* semantically identical changes in a program computing a numerical solution to a mathematical problem, for a given numerical precision, can nevertheless influence numerical results, not speaking of programming errors. One thinks for instance of the compiling options used, the compiler suite chosen, the number of processors used in a parallel implementation, or underlying hardware or operating system,

Indeed, numerical operations in a computer are not associative, therefore the final result might depend on the order of execution of these operations, which is unpredictible on a parallel computer or a GPU. Besides, IEEE standards for the floating point representation of real numbers and operations are not complely respected when high-performance compiler options are chosen, such as when the floating point division is replaced by an inverse multiplications, or power functions or square roots are approximated. Some processors such as the Intel family do not respect the IEEE standard by default (they actually exceed it) unless explicitely specified in the compiler options, with a performance penalty.

Therefore, a comparison of output files of numerical programs on various installations will always lead to some discrepancies, especially in the last digits of the results, even if the code and the inputs are unchanged. Regression tests after a code change for performance reasons, or an addition of features, based on a straightforward comparison of results will always detect bugs, when a manual inspection will show that bugs have not been introduced in the sources.

Except for manual inspection and plotting of results, it is hard to quantify if the results of a numerical code have changed in a significant way when some change has been applied in the process leading to the production of those results. This is especially true for codes using iterations in imaginary time to find stationary solutions of partial differential equations. In this case, the number of iterations to achieve a given numerical precision can vary when some parameter is changed, but nevertheless the end

result is essentially the same if no mistake has been made in the coding process, up to a few digits in the last part of the numerical representation. Only a specific extraction of the results of the last iteration, along with a problem-specific distance criterion in between the reference result and the instance being tested, can give a satisfying automatic measure of the quality of the numerical result. Nevertheless, such an extraction is not automatic in the sense that it is problem-dependent.

Besides, in some application domains of numerical computations such as the aeronautical world, safety rules ask that the results of a specific code, for instance in finite elements calculations, can be reproduced bit for bit. This can be tricky considering that even in a space of a few years many unsuspected parameters can change in a computer system ; some authors have even proposed that virtual machines should be provided with a full environment and run on emulators in order to ensure full reproductibility of numerical experiments.

Some vendors such as Intel now provide libraries that guarantee numerical reproducibility of results, in exchange for a performance penalty.

In this paper we discuss a method to automatically estimate the changes in the output of a numerical code, without having to define problem-dependent metrics, or having to dig too deep in the hardware or software used, for instance by changing the rules of floating point operations in order to guarantee a non-changing result, as some authors propose, or by doing statistical analysis of results, which can be problem-dependent and time consuming.

II. Method

We implemented a simple and universal method which, given two sets P and Q of results of a numerical code, estimates the so-called Kullback-Leibler [1] divergence D from cross-entropies $D = H(P,Q) - H(P)$. This formula is not symmetrical under exchange of P and Q, but could be easily symmetrized, without bringing much more information. If the two sets of results are identical, D will be zero.

The present approximate implementation for this estimation consists in the computation of

$$D_e = \frac{S(C(P+P)) - S(C(P+Q))}{S(C(P+P))}$$

where S is the file size obtained by using the C nondestructive compression program. Here $+$ is the file concatenation operator and $-$ is the traditional substraction. For several well-known compression programs C such as bzip2, gzip,compress, or zip, the information entropy in the compressed file is indeed strongly increased, and the reduction in file size after compression roughly reflects the information entropy of the original file.

We think that using a compression program helps to create an algorithmic expression of the contents of the output of the numerical code, even if the output, as a text file, contains varying information when for instance the compiler options are changed, the compiler changes, or the number of iterations changes, since the results are essentially similar from one iteration to the next and compress very efficiently with modern file compression programs. Many programs used in physics or chemistry indeed only output one text file from which various numerical results have to be extracted after the run, depending on the need, and those results are mixed with various informations such as iteration number, convergence criteria, parallelization choices, etc, which are fairly repetitive and thus are well suited to nondestructive file compression programs. The same goes for numerical results, in which we found a quite high potential for compression, since they do not change significally from one small step iteration to the next, and exhibit many similarities in a given file even if the number of iterations to achieve convergence differ.

III. Results and discussion

First, we will discuss some of the numerical optimizations which we noticed induced

slight changes in numerical results of a typical large-scale numerical project, namely the so-called PW-TELEMAN effort on which we have been working for a number of years in an international collaboration [2], [3]. This code has been recently open-sourced in a software forge on http://www.pw-teleman.org where the present implementation is also avalaible in the form of scripts. This code solves the time-dependent functional theory equations in real time on a regular numerical grid, within the local approximation in time and space, which generates a set of non-linearily coupled one-particle Schrödinger equations, to be first solved to find a stationary solution, which is then submitted to a time-dependent perturbation, allowing to compute a wealth of observables such as electronic emission currents.

A. Examples of optimizations changing numerical results

The first step consisted in optimizing the FFT routines which are used to compute the kinetic energy of the electrons by going to reciprocal space and also to solve Poisson's equation. Historically the NETLIB library was used [4]. In 2012, a move to the FFTW3 library [5], optionally coupled to the Intel MKL library, has been successfully implemented, and has allowed a typical speedup of 50 %. A second effort, done in parallel to the previous step, was devoted to a further parallelization of the code. A MPI parallelization already existed since more than 15 years, with a distribution of propagation of the wave functions over the processors and a reduction of the density in order to compute the coulombic potential as well as the exchange-correlation potential by sharing the grid across the processors. An implementation in OpenMP in the same spirit has been also achieved in 2012 by P.-G.Reinhard. However, the migration from FFTW3 to FFTW3-OMP does not bring a significant speedup of the calculations. Another implementation, using MPI instead, has been done on the grid points, on top of that done on the wave functions. Indeed, the size of the computation box is one of the major bottlenecks of our code. One can occasionally uses about 250 000 grid points, but the standard is more close to a million, and sometimes even more than a million and a half. Meanwhile, the CPU time almost linearly scales with the number of points. This is why in 2013, a MPI parallelization on the grid points has been implemented. The speedup compared with a serial calculation, using FFTW3, ranges from 2 (for 24 wave functions, box size of 963, and 16 processors) to 15 (for 240 wave functions, box size of 963 and 48 processors). There is of course a compromise between the number of processors, the box size and the number of wave functions to be found. Indeed, increasing the number of processors does not systematically enhance the speedup because one then increases the number of communications between the processors.

During the past two years, the use of GPU cores has also been tested, first by using ready-made versions of FFTW and linear algebra routines (CULA). With one GPU core per CPU core, the typical gain is about 1.5 compared to the FFTW3 version. We then started to reprogram in the CUDA language the calculation of the exchange-correlation potential and of the multipoles used in the FALR method to compute the coulombic potential, as well as various field transform operations. On a mixture of cases, the typical speedup went to a factor of 3, using either a combination of cheap CPUs and GPUs (GTX280) or top of the range ones such as K20. A further speedup to a factor of 4-5 was achieved using asynchronous memory transfers in order to increase the utilization of GPUs, the constant memory transfers bringing a huge performance penalty, and finding the optimal register size and block sizes for each GPU (typically, 72 registers on a K20, block sizes of (64,1,1) for 3D grid operations and (512,1,1) for FFTs).

When using a MPI parallelization on the wave functions on top of the use of GPUs (one GPU per core), a speedup by a factor 3 vs

# proc	Compiler	FFT	Multith	Multipr	#procs	Compiler2	FFT	Multith2	Multipr	Divergence
8	intel	CUDA	none	.MPI	16	intel	CUDA	none	.MPI	-0,00347
16	intel	CUDA	none	.MPI	8	intel	CUDA	none	.MPI	-0,00348
4	intel	CUDA	none	.MPI	8	intel	CUDA	none	.MPI	-0,00550
4	intel	CUDA	none	.MPI	16	intel	CUDA	none	.MPI	-0,00587
4	gfortran	netlib	OpenMP	.seq	2	gfortran	netlib	OpenMP	.seq	-0,01131
2	gfortran	netlib	OpenMP	.seq	4	intel	fftw	OpenMP	.seq	-0,05539
1	intel	CUDA	OpenMP	.seq	8	intel	netlib	none	.MPI	-0,05541
1	gfortran	fftw	none	.seq	1	intel	CUDA	OpenMP	.seq	-0,05543
4	gfortran	netlib	OpenMP	.seq	4	intel	fftw	OpenMP	.seq	-0,05547
8	intel	CUDA	none	.MPI	1	gfortran	fftw	OpenMP	.seq	-0,11420
8	intel	CUDA	none	.MPI	4	gfortran	fftw	OpenMP	.seq	-0,11420
16	intel	CUDA	none	.MPI	4	gfortran	fftw	OpenMP	.seq	-0,11433
16	intel	CUDA	none	.MPI	16	gfortran	fftw	OpenMP	.seq	-0,11472
16	intel	CUDA	none	.MPI	1	gfortran	fftw	none	.seq	-0,11472
8	intel	CUDA	none	.MPI	16	gfortran	fftw	OpenMP	.seq	-0,11497

Fig. 1. Typical results on the PWTELEMAN library for various options

the pure MPI case on CPUs is also attained because the asynchronous transfers are very hard to coordinate in this case.

All possible compilations options and available compilers (here, the latest versions of GFORTRAN and Intel Fortran) are combined with a shell script, generating a set of executable files, which are then run on the specified examples, with a varying number of processors and GPUs.

B. Discussion of the results

We present on figure 1 the results of a typical analysis of output files with the presently discussed method, for the same input file, namely a carbon dimer represented on a 128^3 cubic grid of points separated by 0.4 a.u, inducing first the computation of the ground state, then 1000 time steps. The corresponding input file is given as C2-128 in the samples of the distribution of PWTELEMAN. All the combinations of options and a ranging on the number of processors used gave rise to more than 1000 results ; here we give the results for the lowest values of the divergence, close to average values (5 percent), and the higher values obtained. We used the latest version of bzip2 as a compression program.

The results indeed show that changes in compiler options or in the number of processors will result in a value of D_e of a few percent, reflecting the slight changes in the numerical results, due to the noncommutativity of numerical operations, which hopefully are limited to the last digits. Various messages in the output of the programs also change, explaining the small change in the results. If the compiler is changed, going from the latest GFORTRAN implementation to the commercial Intel suite of compilers, or the parallelization methods (MPI/OpenMP) or the way the FFT is computed (NETLIB/FFTW3 library / CUDA) those results change to about ten percent. Reassuringly, the use of GPU leads to a value of D_e of less than one percent on monoprocessor systems. Programming bugs that either lead to no executable file, to a program failing to reach completion, or to a notable change in results will be immediately detected by a value of D_e far exceeding this ten percent value. This allows for an automatic quantification of regression in the code when a change is made in the sources, discarding small numerical artefacts related to performance increases or variations in systems architecture, be it hardware or software relard. A threshold has to be defined first by manual inspection of the results, but we are confident that a fifteen percent rule is reasonable, which we could try by voluntarily or not introducing bugs in the code base, which immediatly result in a divergence of fifty percent or more of the results.

IV. Conclusion

In the present paper, we have demonstrated in the case of a typical numerical code that an approximate method to compute the Kullback-Leibler divergence of results after changes are made in the production process of those results can help automatically estimate the reproducibility of numerical results, even if the compiler, the hardware, the parallelization method is changed. This way, bugs can be detected without having to define problem-dependent, specific metrics for the changes in numerical results, or to spend too much effort in arcane hardware, numerical, or software problems in order to ensure bit for bit reproducibility.

ACKNOWLEDGMENT

The author would like to thank GENCI and CRIHAN for computational time (projects projects x2014096171 and 007 respectively), and all the indirect contributors to the present paper, especially K.Brymora, D.Brusson, P.-G.Reinhard, E.Suraud, M.Dinh, as well as many other contributors to the PWTELEMAN library.

REFERENCES

[1] S. Kullback and R. A. Leibler, "On Information and Sufficiency," *The Annals of Mathematical Statistics*, vol. 22, no. 1, pp. 79–86, 1951. [Online]. Available: http://dx.doi.org/10.1214/aoms/1177729694

[2] F. Calvayrac, P.-G. Reinhard, E. Suraud, and C. Ullrich, "Nonlinear electron dynamics in metal clusters," *Physics Reports*, vol. 337, no. 6, pp. 493–578, 2000.

[3] J. Wang, C.-Z. Gao, F. Calvayrac, and F.-S. Zhang, "Collision dynamics of proton with formaldehyde: Fragmentation and ionization," *The Journal of Chemical Physics*, vol. 140, no. 12, p. 124306, Mar. 2014. [Online]. Available: http://scitation.aip.org/content/aip/journal/jcp/140/12/10.1063/1.4868985

[4] P. Swarztrauber, "Vectorizing the FFTs," in *Parallel Computations*, G. Rodrigue, Ed. New York: Academic Press, 1982, pp. 51–83.

[5] M. Frigo and S. G. Johnson, "The design and implementation of FFTW3," *Proceedings of the IEEE*, vol. 93, no. 2, pp. 216–231, 2005, special issue on "Program Generation, Optimization, and Platform Adaptation".