Good Practices for Reproducible Research

Luka Stanisic and Arnaud Legrand

MESCAL/LIG/Univ. of Grenoble, Grenoble, France
firstname.lastname@imag.fr

1. Introduction

In the last decades, computers, operating systems and software running on it have reached such a level of complexity that it has become very difficult (not to say impossible) to control them perfectly and to know every detail about their configuration and operation mode. As a consequence, it becomes less and less reasonable to consider computer systems as deterministic, especially when measuring execution times or performance of large distributed computing systems. These systems have become so complex that it is almost impossible to fully understand their behavior, making the full reproduction of measurements extremely difficult. To some extent, although these systems have been designed and built by humans, studying modern computers has become very similar to studying a natural phenomenon. Just like most other scientific domain, many of the conclusions are based on experiment results and their analysis. Nevertheless, in many computer science articles, the experimental protocol is rarely detailed enough to allow others to reproduce the study and possibly build upon it. This is all the more surprising as reproducibility and falsifiability are yet the basis of modern science as defined by Popper.
In the recent years, the more and more frequent discovery of frauds or mistakes in published results has shed the light on the importance of reproducible research in computational sciences. Many tools partially addressing these issues have thus been proposed in different fields (biology, image processing, etc.) and although there is an urgent need for changing practices in computer science, how we should proceed is not yet clear. We have recently written an article [3] that contains a completely replicable analysis of the large set of experiments. We took care of doing these experiments in a clean, coherent, well-structured and reproducible way that allows anyone to inspect how they were performed.

In this presentation, we propose to explain and demonstrate the basics of our workflow and the technology we used.

2. Context

The reproducible research [3], of which we will illustrate the internals, was done in the context of High-Performance Computing (HPC). Modern HPC infrastructures are typically made of hybrid machines with several CPUs and GPUs. To efficiently exploit all these heterogeneous resources, programmers have to explicitly allocate computation-intensive kernels and manage data transfers between the different processing units. It is common to rely on task-based runtimes that provide the right abstraction to efficiently perform such tedious management and optimizations. Designing and configuring such runtime systems is itself a challenging problem for which we believe the systematic use of modeling and simulation can provide an answer. To support our claim, we have ported StarPU [1], a dynamic task-based scheduling runtime for hybrid architectures, on top of the SimGrid simulator [2] and we have shown that this combination allowed to provide accurate performance predictions for dense linear algebra kernels.
To validate our approach and our models, we had to perform measurements on several, sometimes not dedicated, machines, and work on complex beta code that often needed to be modified to fit our needs. In this context, even the smallest misunderstanding or inaccuracy about a parameter at small scale often results in a completely different behavior at the macroscopic level. It was thus crucial to carefully collect all the useful metadata, to use a well-planed experiment design and a coherent analysis, all in order to be able to easily reproduce the experiment results.

3. From Experiments to Article: Reproducibility vs. Replicability

We can distinguish between two main trends in reproducible research. The first one aims at completely automatizing the whole process (experiments, analysis and the final article), allowing others to replicate it. This approach is particularly popular in computational sciences. However, maintaining such process functional in time and portable across platforms is quite burdensome. Additionally, it imposes restrictions on what can be done, it is difficult to extend and hence not always usable on a daily basis. Furthermore, focusing on automatic replication hides information on why and how things were done. The second approach is much lighter and relies on literate programming of code and analysis scripts (i.e. documenting why and how to use the code) and on the systematic use of recipes (i.e., how to build environments). These two approaches apply to both experiments and analysis part of the research.

3.1. Reproducible Experimentation

When measuring execution time of a complex application on a modern computer, two runs in a row with the same setup rarely provide the exact same timing in nanoseconds. Such experiments are de facto not *replicable*, but only *reproducible* provided that enough details about the configuration are available. We have thus taken a particular care to ensure that all our experiment results are reproducible and will describe in our presentation the most important ones.

Before each measurement, we **automatically log information** about the machine, such as the current CPU frequency and governor, the memory hierarchy, the versions of Linux and gcc, etc. We also **systematically recompile** software before using it to conduct experiments, and keep all the configuration and compilation outputs. Additionally, we enforce that all changes to the source code are committed in the **revision control system** and we keep track of the hash of the Git/SVN version of the source code. This enables **provenance tracking**, i.e, to know which results were obtained with which code, so they can be later easily investigated, compared or reproduced. Finally, we save the measurement results (makespan, GFlop/s rate) together with execution traces. During the development period, workflow and data format went through several changes and adjustments and we used a **laboratory notebook** to keep track about all modifications.

3.2. Replicable Analysis and Article

Unlike experimentation, analysis is only concerned with the final output (figures, tables, numerical summaries) and not with the time it takes to compute it. Therefore, for a given experimental data set, results of the analysis should always remain the same, which makes them by essence *replicable*. Reproducible research tools and technologies are still in their infancy and there are many different alternatives (Rstudio/knitr, ActivePapers, Elsevier executable papers, etc.). We tried several of them and we have finally chosen to use **org-mode**, more precisely org-babel, that allows to **combine multiple processing and analysis codes** as well as their **output results** inside a plain text file. Compared to other alternatives, we feel that org-mode is mature, flexible and lightweight enough for both a daily usage and to allow to produce high quality reproducible articles.

The article we have written **combines within a single plain text file the body of the article along with all the analysis**, all of which are later exported into standard pdf document. It has a hierarchical structure, with different types of code, including Shell (to manipulate data files), R (for plotting figures) and LaTeX (to finely control formatting details). We made **all raw data and traces publicly available** on figshare [4]. To replicate the article, all these data should be downloaded and unpacked. Then all useful information is extracted into csv files by parsing and filtering raw data files that contain many additional metadata. Finally, R is used to load, process and plot data, ensuring that all figures are consistent throughout the whole article. This way, **our experimental and analysis results can be inspected, referenced, or even reused** in any other research project.

4. Conclusion

The approach we propose to present was very effective for doing reproducible research in our context. We strongly believe that it can be easily applied to many other situations and that presenting such methodology can be beneficial to the audience.

References

1. Cédric Augonnet, Samuel Thibault, Raymond Namyst, and Pierre-André Wacrenier. StarPU: A Unified Platform for Task Scheduling on Heterogeneous Multicore Architectures. *Concurrency and Computation: Practice and Experience, Special Issue: Euro-Par 2009*, 23:187–198, February 2011.
2. Henri Casanova, Arnaud Legrand, and Martin Quinson. SimGrid: a Generic Framework for Large-Scale Distributed Experiments. In *proceedings of the 10th IEEE International Conference on Computer Modeling and Simulation (UKSim)*, April 2008.
3. Luka Stanisic, Samuel Thibault, Arnaud Legrand, Brice Videau, and Jean-François Méhaut. Modeling and Simulation of a Dynamic Task-Based Runtime System for Heterogeneous Multi-Core Architectures. In *Proceedings of the 20th Euro-Par Conference*. Springer-Verlag, August 2014.
4. Companion of the StarPU+SimGrid article. Hosted on Figshare, 2014. Online version of this article with access to the experimental data and scripts (in the org source); http://dx.doi.org/10.6084/m9.figshare.928095.