SF2A 2016
C. Reylé, J. Richard, L. Cambrésy, M. Deleuil, E. Pécontal, L. Tresse and I. Vauglin (eds)

PERSPECTIVES IN NUMERICAL ASTROPHYSICS: TOWARDS AN EXCITING FUTURE IN THE EXASCALE ERA

V. Reverdy[1]

Abstract. In this discussion paper, we investigate the current and future status of numerical astrophysics and highlight key questions concerning the transition to the exascale era. We first discuss the fact that one of the main motivation behind high performance simulations should not be the reproduction of observational or experimental data, but the understanding of the emergence of complexity from fundamental laws. This motivation is put into perspective regarding the quest for more computational power and we argue that extra computational resources can be used to gain in abstraction. Then, the readiness level of present-day simulation codes in regard to upcoming exascale architecture is examined and two major challenges are raised concerning both the central role of data movement for performances and the growing complexity of codes. Software architecture is finally presented as a key component to make the most of upcoming architectures while solving original physics problems.

Keywords: numerical astrophysics, exascale, cosmology, simulations

1 Introduction

Over the last decades, numerical simulations have progressively become a central element of modern astrophysics. The fact that astrophysical processes are often difficult to reproduce in laboratories and the fact that these processes are often complex multiscale multiphysics problems have helped numerical simulations to acquire this role. In the meantime, astrophysics has become together with climate science (Lapenta et al. 2013; Towns et al. 2014), one the biggest user of supercomputing resources. However, again and again, the same pattern keeps happening in numerical astrophysics: 1) To answer a specific question, a numerical project is designed. 2) Because existing codes do not allow to answer the exact question, a new physics module needs to be developed. 3) But as parallelization techniques are often complex to implement, a lot of work needs to be put into making the physics module compatible with the existing code architecture. 4) However, as the project has been defined as a physics project and not as an applied mathematics or computer science project, the simplest implementation technique is chosen: the code is forked into a specific version dedicated to the problem, and the new physics is integrated thanks to quick and dirty hacks. 5) Because astrophysical processes often involve multiphysics aspects or high-dimensional parameter spaces, simulation are run and their parameters are progressively adjusted to fit observational results. 6) The conclusion is generally that a set of parameters fitting the observational data can be found, but answering the original question would require more resolution or more data points. 7) A paper is published presenting these conclusions, but because the code has been implemented through quick and dirty hacks, the authors judge that their code is not mature enough to be publicly released on a open-source license, therefore preventing any possibility of cross validation by a larger community. 8) To investigate the problem with more resolution and more data points, more funding is requested, the process restart at step 1), and the code is forked again. Even though the steps we just listed are intentionally exaggerated, some elements of this list can be found, at least partially, in many large scale simulation projects. As the numerical astrophysics community is now aiming for exascale simulations in the coming years (Berczik et al. 2013), and as porting codes to the exascale will be a major challenge involving a lot of resources and time (Dongarra et al. 2011), we offer a quick review of some of the questions numerical astrophysics will have to face while trying to ensure original, high-quality, reliable, and reproducible science results.

[1] University of Illinois at Urbana-Champaign, Department of Astronomy, MC-221, 1002 W. Green Street, Urbana, IL 61801

2 On astrophysics simulations

2.1 What is an astrophysics simulation? Why running high performance simulations?

Before going any further one has to define what an astrophysics simulation is. Is there any conceptual difference between solving a single equation on a computer, fitting data to compute model parameters or to produce mock catalogs, quickly exploring or test an hypothesis using a script or a high-level language, and running high performance simulations on supercomputers? Technically, all these tasks boil down to solving a problem algorithmically using a Turing machine. However, they differ vastly in terms of motivation and implementation.

In the first case, computers are just used as powerful calculators to get a numerical value from solving a symbolic equation. In the second case, when we use computers to fit data and produce mock catalogs, are we actually simulating anything? The question is worth asking because a significant fraction of present-day simulations are run to produce mock catalogs to prepare observational surveys and check experimental data (Lowing et al. 2015; Fosalba et al. 2015). But fitting and producing mock catalogs that emulate real observations, does not require any understanding of the underlying physics derived from its firsts principles. Trained properly on large existing data sets, machine learning algorithms and neural networks are perfectly suited to produce mock catalogs (Xu et al. 2013; Kamdar et al. 2016) while using far less computing resources, and being sometimes more accurate than high performance numerical simulations. As they will become more and more widely used, and as they may outperform traditional simulations to mimic observational and experimental results, machine learning techniques are likely to raise interesting epistemological questions regarding the very notion of understanding in science.

Then how do the third and fourth case described in the first paragraph differ? Present-day personal computers and modern high-level languages allow to quickly prototype, design and implement simulations to test hypotheses on toy models. These simulations are generally very relevant to explore analytical models and provide qualitative results about them. But if personal computers can allow to investigate analytical models, and if machine learning can provide very accurate fitting solutions to mimic real observational results, why are we still doing high performance simulations in 2016? We argue that one of the main point of high performance simulations is to investigate the emergence of complexity from fundamental principles (Anderson et al. 1972) to: 1) test these principles, 2) verify how less fundamental models can be derived from first principles, and 3) understand how complex physical processes actually work. According to this view, the goal of high performance simulations is not to mimic what we observe, but to understand how the fundamental laws of physics are connected to what we observe. Increasing the number of free parameters in simulations increase the likelihood of being able to reproduce observations, but it provides no guarantee that the underlying physics is correct. On the contrary, gaining in abstraction following a traditional reductionist approach would allow to simulate a wider range of astrophysical processes from a smaller set of laws, and would limit the problem of degenerated solutions.

2.2 Why more computing power?

As noted in the introduction, numerical astrophysics is currently on a race towards more and more computing power. The first petascale supercomputer was tested in 2008 (Barker et al. 2008), and we are likely to enter in the exascale era during the next decade. But why more computing power is even required? Common motivations for more computing power can fit into one of two categories. The first category corresponds to the case in which the level of physics complexity remain the same but either larger simulations in time or in space, more resolution, or more simulations to improve statistical accuracy are required (Alimi et al. 2012). In that case, code scalability is of primary concern. The second category corresponds to the case in which more complex problems need to be solved either regarding the number of physical phenomena that has to be taken into account (Vogelsberger et al. 2014), or regarding the geometry of the problem. In that case, as entirely new modules often need to be developed, genericity and modularity of software architectures is of primary concern.

But there is a third category: using more computing power to gain in abstraction and genericity while reducing the number of free parameters. For example, in numerical cosmology, over the years, several techniques have been developed to analyze light propagation: some have been focusing on weak lensing (Jain et al. 2000), some others on the late-time integrated Sachs-Wolfe effect and some others on the accurate computation of redshifts in cosmological simulations. However, all these effects do not need separate modelling as they all originate from the same phenomenon: the propagation of photons according to general relativity. Directly integrating the geodesics equations instead of implementing completely different approaches depending on the

effect allow to use a single code for a wide range of analyses while reducing the parameter count and improving the accuracy of results (Reverdy 2014).

Over the years, to cover more and more physics domains, and to target more and more platforms, astrophysics codes have grown to an unprecedented scale. Because this growth has often been the result of an unplanned community effort, it has lead to an increase of code complexity. Adding new physics aspects to these codes therefore requires a deep knowledge of the implementation including data structure, algorithms, and parallelism, and often involves far more work on the computational and technical aspects than on the implementation of solvers and actual physics equations. On the top of the above-mentioned advantages, gaining in abstraction also offers the opportunity to drastically reduce code complexity, making further developments easier, more modular, and more maintainable in the long run.

3 Challenges for the exascale era

As already pointed out, as the computing power of supercomputers keeps growing, an exascale supercomputer is very likely to become reality within the next decade, regardless of the readiness status of astrophysical codes. In the next sections, we discuss two major challenges astrophysics codes will face in their race to the exascale.

3.1 Pure computing power and data movement

Very few present-day codes are able to fully leverage petascale supercomputers. One of the main reason is that most of these codes inherited from codes that were designed in the 2000's (Teyssier 2002; Springel 2005) with computing power as a primary focus. The democratization of multithreading has deeply affected supercomputing, and a major effort has been put into exploiting efficiently both inter and intra-node parallelism. Even though hybrid MPI/OpenMP or MPI/pthread parallelism is more common in 2016 than in 2006 (Bryan et al. 2014), not all parallel codes exploit these techniques. But this is not the main reason why petascale supercomputers are underused. While numerical astrophysics was focusing on pure computing power, the main bottleneck of high performance computing was silently moving away from it. A quick comparison between the 2004 and 2014 world's top supercomputer according to the TOP500 ranking lead to the following numbers: while the pure computing power has grown by three orders of magnitude in a decade, the total memory available on these systems has grown by two order of magnitude, less than two orders of magnitude for the storage system, and even less when it comes to compare cache and random access memory speed. Unless a revolutionary memory technology emerges in the coming years, leveraging supercomputers will rely more and more on how to feed processing units with a constant flow of data (Reed & Dongarra 2015). In other words, the pure computing power is not the problem anymore, the problem is how to store and efficiently access data. And most astrophysics codes were not originally designed with this problem in mind.

On modern platforms, L1 cache access is roughly two orders of magnitude faster than RAM access. Therefore when a CPU needs elements that have not been put in cache due to complex access patterns or complex branching strategies, hundreds of cycles can be lost for doing nothing except waiting. Making the most of current and upcoming architectures involve improving cache friendliness and data access patterns for both optimizing data transfer between random access memory and central processing units, and facilitating vectorization and the use of SIMD instruction sets. In that regard, data structures have a major role to play (Hartmann et al. 2008), as they often represent the backbone of simulation codes. These codes will not be able to scale up to the exascale without solving issues regarding the storage of massive amount of data with predictable access patterns.

3.2 Code complexity

The second major challenge simulation codes will face concern their growing complexity. This complexity arises from the conflict between the need of low-level development for optimization purpose and the need of high-level development to incorporate more and more numerical methods, algorithms, solvers, and physical phenomena. Because most of these aspects are orthogonal to each other, one should be able to compose them to design their code. But how to achieve modularity, genericity and efficiency at the same time? For now, many of the most used codes in numerical astrophysics focus on a particular set of physics problems, with a particular set of numerical methods and target specific architectures. Moreover, all aspects of the computation are very intertwined. One of the direct consequences of this intertwining is encountered by many scientist: slightly modifying the physics of an existing code often involves weeks or months of technical work, because the physics cannot be easily changed without impacting the entire code.

If the hardware has changed a lot over the last decades, programming approaches have, on the contrary, and for the most part, stayed the same. Developing a script to visualize data and compute some statistics on a personal computer is not the same as developing high performance libraries involving years of effort. In the second case, the choice of a programming language, the choice of a particular design or interface, and even the choices of function signatures including function names and arguments are everything but technical details. Original architecture decisions can have tremendous impacts on the overall design years later. This can be illustrated through the design of something as simple as a data cube, in other words a cube defining a region of a discretized space and containing values of physical quantities. At first, the problem seems simple: a cube can be defined through the coordinates of its center and its edge length. But soon, design questions arise: How to store coordinates? How to manage the precision on the coordinates? Should fixed-point be preferred to floating-point representation to avoid non homogeneous precision around 0? Can the number of dimensions be made arbitrary? In that case, how to compute the number of cells, faces, edges, and vertices recursively at compile-time? Should the cube be also stored as the coordinates of its vertices so that cells, faces and edges can be referenced recursively as independent objects built on top of the same underlying vertices? Should the coordinates be allocated on the stack or on the heap? Regarding to the storage of an array of data cubes, can the array of structures be converted into a structure of arrays to provide better vectorization opportunities? How to deal with different system of coordinates and affine transformation? Do data cubes own physical quantities or do they contain links to them? etc... In the end, designing a data cube happen to be a very complex problem, with fundamental consequences in terms of genericity, ease-of-use and efficiency. When a simulation stores billions of data cubes to manage space discretization, all these design decisions can have a major impact on the overall performances and memory usage of a code. In the end, investigating complex physics problems in the exascale era will require modular codes because otherwise, adjusting the slightest element will require months of development and make exascale codes unmaintainable.

4 Conclusions

As we illustrated throughout this paper, designing astrophysics codes capable of leveraging exascale architectures will not be an easy task. Designing codes to answer original questions instead of simply aiming at improving resolutions will be even more complicated. However, investigating the emergence of complexity from fundamental laws using extra computing power offers plenty of room for originality. In that context, software architecture have a central role to play to achieve, at the same time, genericity, efficiency and ease-of-use. Accomplishing this task will require, at least, astrophysics, computer science, applied mathematics and software engineering aspects. But because all these aspects are interrelated, and because the design of the most fundamental software components can have drastic implications in the long run, only a transdisciplinary – not a multidisciplinary – approach will succeed.

Vincent Reverdy has been supported by the National Science Foundation Grant AST-1313415.

References

Alimi, J.-M., Bouillot, V., Rasera, Y., et al. 2012, The International Conference for High Performance Computing, Networking, Storage and Analysis, Salt Lake City, Utah, USA, 10-16 November 2012

Anderson, P. W. et al. 1972, Science, 177, 393

Barker, K. J., Davis, K., Hoisie, A., et al. 2008, in Proceedings of the 2008 ACM/IEEE Conference on Supercomputing, SC '08 (Piscataway, NJ, USA: IEEE Press), 1:1–1:11

Berczik, P., Spurzem, R., Zhong, S., et al. 2013, Up to 700k GPU Cores, Kepler, and the Exascale Future for Simulations of Star Clusters Around Black Holes, ed. J. M. Kunkel, T. Ludwig, & H. W. Meuer (Springer), 13–25

Bryan, G. L., Norman, M. L., O'Shea, B. W., et al. 2014, ApJS, 211, 19

Dongarra, J. et al. 2011, International Journal of High Performance Computing Applications, 25, 3

Fosalba, P., Crocce, M., Gaztañaga, E., & Castander, F. J. 2015, MNRAS, 448, 2987

Hartmann, J., Krahnke, A., & Zenger, C. 2008, Applied Numerical Mathematics, 58, 435

Jain, B., Seljak, U., & White, S. 2000, ApJ, 530, 547

Kamdar, H. M., Turk, M. J., & Brunner, R. J. 2016, MNRAS, 455, 642

Lapenta, G., Markidis, S., Poedts, S., & Vucinic, D. 2013, Computing in Science and Engineering, 15, 68

Lowing, B., Wang, W., Cooper, A., et al. 2015, MNRAS, 446, 2274

Reed, D. A. & Dongarra, J. 2015, Commun. ACM, 58, 56
Reverdy, V. 2014, PhD thesis, Paris Observatory
Springel, V. 2005, MNRAS, 364, 1105
Teyssier, R. 2002, A&A, 385, 337
Towns, J., Cockerill, T., Dahan, M., et al. 2014, Computing in Science and Engineering, 16, 62
Vogelsberger, M., Genel, S., Springel, V., et al. 2014, MNRAS, 444, 1518
Xu, X., Ho, S., Trac, H., et al. 2013, ApJ, 772, 147