Good Practices for Reproducible Research

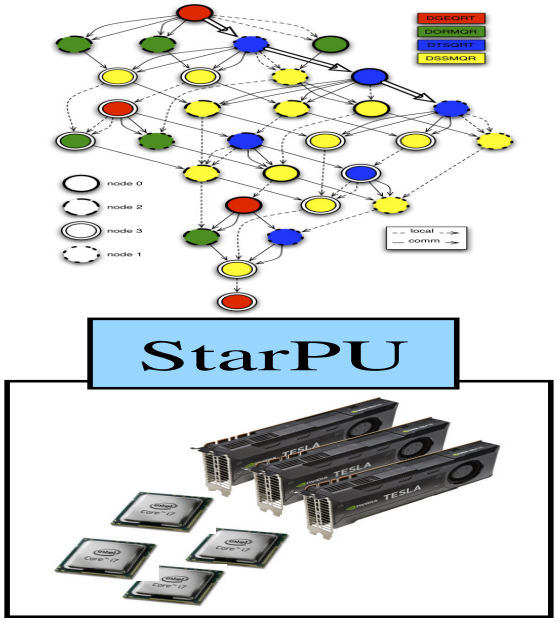
Luka Stanisic Arnaud Legrand

CNRS/Inria/University of Grenoble, France

Atelier en Evaluation de Performances, Sophia-Antipolis
June 13, 2014

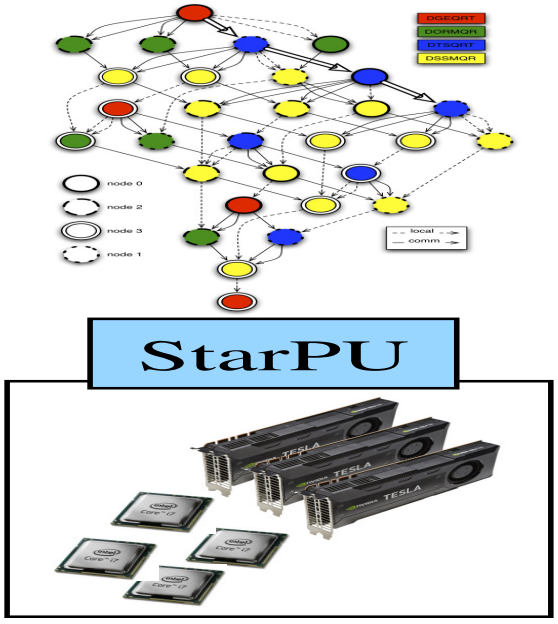
Context

- Hybrid machines with both multi-core CPUs and GPUs are now commonplace
- Portable performances across architectures is extremely challenging $\rightsquigarrow$ adaptive task-based runtime (StarPU,DAGuE, KAAPI, ...)



Context

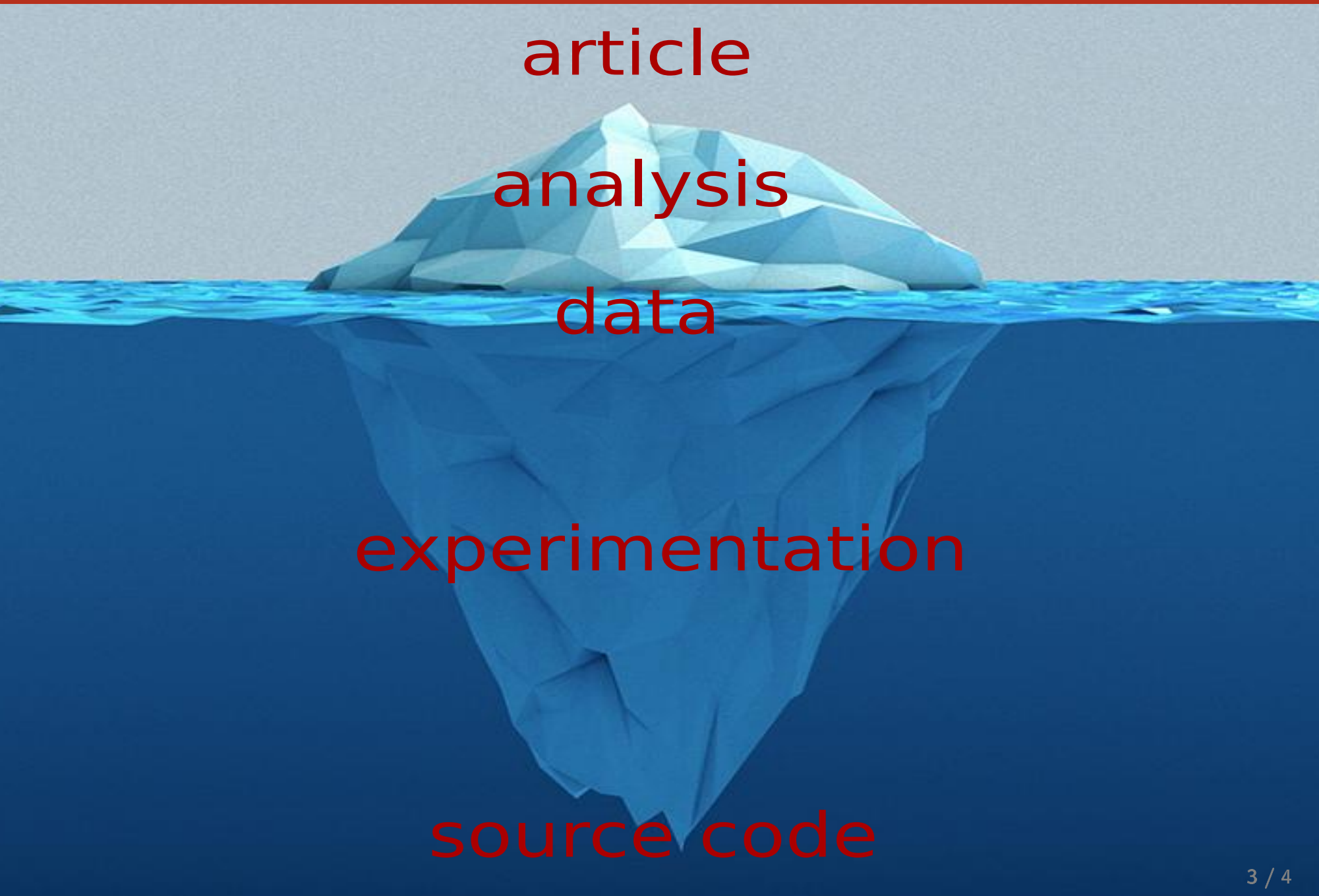
- Hybrid machines with both multi-core CPUs and GPUs are now commonplace
- Portable performances across architectures is extremely challenging $\rightsquigarrow$ adaptive task-based runtime (StarPU,DAGuE, KAAPI, ...)



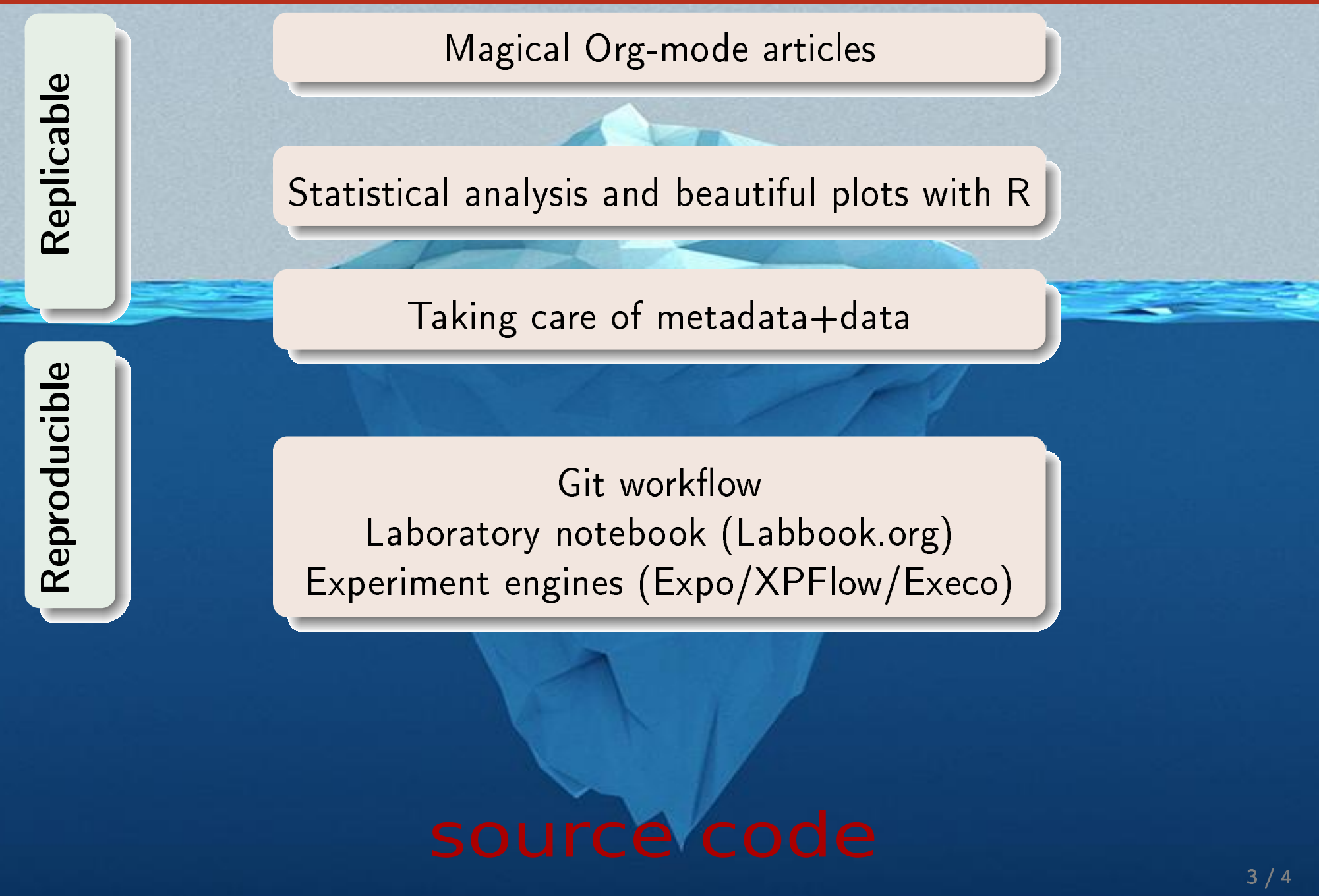
Conducting Experiments and Reporting Results

- Prototype code + fragile machine configuration $\rightsquigarrow$ results are hard to reproduce
- Parameter/algorithm modification (granularity, scheduling, application structure, ...) can have a huge impact on performances
- Making sure new feature work on a wide variety of setups

Reproducible Research



Reproducible Research



A Reproducible Article

Modeling and Simulation of a Dynamic Task-Based Runtime System for Heterogeneous Multi-Core Architectures

Luka Stanisic[1], Samuel Thibault[2], Arnaud Legrand[1], Brice Videau[1], and Jean-François Méhaut[1]

[1] CNRS/Inria/University of Grenoble, France, firstname.lastname@imag.fr
[2] University of Bordeaux/Inria, France, samuel.thibault@labri.fr

Abstract. Multi-core architectures comprising several GPUs have become mainstream in the field of High-Performance Computing. However, obtaining the maximum performance of such heterogeneous machines is challenging as it requires to carefully offload computations and manage data movements between the different processing units. The most promising and successful approaches so far rely on task-based runtimes that abstract the machine and rely on opportunistic scheduling algorithms. As a consequence, the problem gets shifted to choosing the task granularity, task graph structure, and optimizing the scheduling strategies. Trying different combinations of these different alternatives is also itself a challenge. Indeed, getting accurate measurements requires reserving the target system for the whole duration of experiments. Furthermore, observations are limited to the few available systems at hand and may be difficult to generalize. In this article, we show how we crafted a coarse-grain hybrid simulation/emulation of StarPU, a dynamic runtime for hybrid architectures, over SimGrid, a versatile simulator for distributed systems. This approach allows to obtain performance predictions accurate within a few percents on classical dense linear algebra kernels in a matter of seconds, which allows both runtime and application designers to quickly decide which optimization to enable or whether it is worth investing in higher-end GPUs or not.

1 Introduction

High-Performance Computing architectures now widely include both multi-core CPUs and GPUs. Exploiting the tremendous computation power offered by such systems is however a real challenge. Programming them efficiently is a first concern, but managing the combination of computation execution and data transfers can also become extremely complex, particularly when dealing with multiple GPUs. In the past few years, it has become very common to deal with that through the use of an additional software layer, a runtime system, based on the task programming paradigm [3,4,7]. Applications are expressed as a task graph with data dependencies, i.e., a Directed Acyclic Graph (DAG), and provide both CPU and GPU implementations for the tasks. The runtime can then schedule

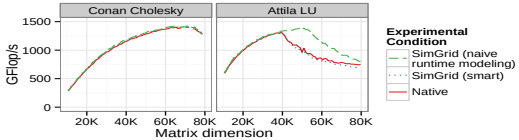


Fig. 1. Illustrating the influence of modeling runtime. Careless modeling of runtime may be perfectly harmless in some cases, it turns out to be misleading in others

of runtime proved to be the size of GPU memory. Such hardware limits force the scheduler to swap data back and forth between the CPUs and GPUs. These data movements saturate the PCI bus, producing a tremendous impact on overall performance. It is thus critical to keep track of the amount of memory allocated by StarPU during the simulation to make sure the scheduler will behave in the same way for both real native executions and simulations.

Figure 1 illustrates the importance of taking into account the runtime parameters described above. Each curve depicts GFlop/s rate of experiments representing 90 different matrix dimensions (matrix dimension 80,000 corresponds to ≈25GB). Solid line *Native* shows the execution of StarPU on the native machine, while the other two are the results of the simulation: *naive* for execution without any runtime adjustments and *smart* with all of them included. The left plot depicts a situation where all these optimizations have very little influence as both *naive* and *smart* lines are almost overlapping with the *native* line. On the other hand, for some other machines and applications (plot on the right), having a precise modeling of runtime is critical as otherwise, simulation may highly overestimate the performance for the larger matrix size. Nonetheless, we remind that the excellent predictions achieved in these examples are also the result of the careful modeling of communications and computations, which we will present in the next Sections.

6 Modeling communication in hybrid systems

Due to the relatively low bandwidth of the PCI bus, applications running on hybrid platforms often spend a significant fraction of the total time transferring data back and forth between the main RAM and the GPUs. Modeling communication between computing resources is thus of primary importance. As a first approximation (see Figure 2(a)), the transfer time between resources could be modeled as a single link with a latency and a transfer rate corresponding to typical characteristics of the PCI bus. However, such modeling does not account for many architectural aspects. First, the bandwidth between CPU and GPU is asymmetrical. Second, communication characteristics are not uniform among all pairs of CPUs and GPUs, as it depends on the chipset architecture. We decided to account for it by using a dedicated uplink and a downlink with different characteristics for each pair of resources (see Figure 2(b)). Furthermore,

http://dx.doi.org/10.6084/m9.figshare.928338