# Perspectives in numerical astrophysics

## Towards an exciting future in the exascale era

Vincent Reverdy

ILLINOIS
UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN

Department of Astronomy, University of Illinois at Urbana-Champaign

SF2A, June 17, 2016

# Introduction

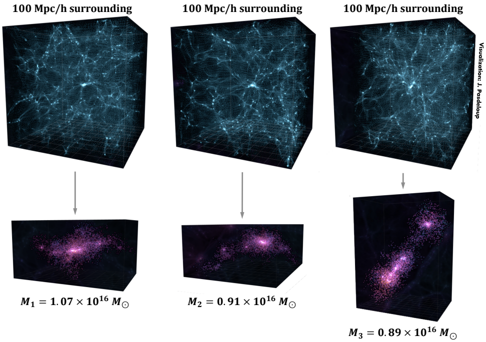
# Outline

# Numerical cosmology

# How did it start?

## Dark Energy Simulation Series: Full Universe Run

- 3 different cosmological models
- Scale of the observable Universe
- Fine-tuned version of RAMSES
- 80 000 cores (Curie Supercomputer)
- 1.6 Petabytes of data



# How did it start?

## Physics lessons

- Newtonian simulations
- How to take into account relativistic effects?
- Kinematic effects (photon trajectory)
- Dynamic effects (backreaction conjecture)

## Numerical lessons

- Codes like RAMSES are difficult to modify: lack of genericity
- Current codes won't scale up to exascale: lack of performance
- Current petascale supercomputers are massively under-used

# Why more computing power?

# Why more computing power?

## Same physical complexity

- Bigger and/or longer simulations
- More resolution (space, time...)
- More simulations (statistical accuracy)

## More complex problems

- More orders in equations
- More physical effects
- More complex geometries

# Why more computing power?

4 August 1972, Volume 177, Number 4047 **SCIENCE**

## More Is Different

Broken symmetry and the nature of the hierarchical structure of science.

P. W. Anderson

The reductionist hypothesis may still be a topic for controversy among philosophers, but among the great majority of active scientists I think it is accepted without question. The workings of our minds and bodies, and of all the animate or inanimate matter of which we have any detailed knowledge, are assumed to be controlled by the same set of fundamental laws, which except under certain extreme conditions we feel we know pretty well.

It seems inevitable to go on uncritically to what appears at first sight to planation of phenomena in terms of known fundamental laws. As always, distinctions of this kind are not unambiguous, but they are clear in most cases. Solid state physics, plasma physics, and perhaps also biology are extensive. High energy physics and a good part of nuclear physics are intensive. There is always much less intensive research going on than extensive. Once new fundamental laws are discovered, a large and ever increasing activity begins in order to apply the discoveries to hitherto unexplained phenomena. Thus, there are two dimensions to basic research. The frontier of science extends all along a long line from the newest and most modern intensive research, over the ex- less relevance they seem to have to the very real problems of the rest of science, much less to those of society.

The constructionist hypothesis breaks down when confronted with the twin difficulties of scale and complexity. The behavior of large and complex aggregates of elementary particles, it turns out, is not to be understood in terms of a simple extrapolation of the properties of a few particles. Instead, at each level of complexity entirely new properties appear, and the understanding of the new behaviors requires research which I think is as fundamental in its nature as any other. That is, it seems to me that one may array the sciences roughly linearly in a hierarchy, according to the idea: The elementary entities of science X obey the laws of science Y.

| X | Y |
|---|---|
| solid state or many-body physics | elementary particle physics |
| chemistry | many-body physics |
| molecular biology | chemistry |
| cell biology | molecular biology |
| ⋮ | ⋮ |
| psychology | physiology |
| social sciences | psychology |

### On the notion of emergence

- Simple rules ⇒ complex phenomena

# Why more computing power?

## Same physical complexity

- Bigger and/or longer simulations
- More resolution (space, time. . . )
- More simulations (statistical accuracy)

## More complex problems

- More orders in equations
- More physical effects
- More complex geometries

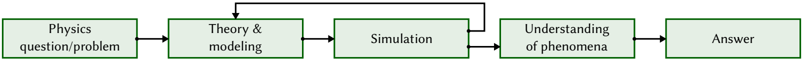
## What about less complex physics?

- Less specific
- Fewer parameters
- More abstract
- More generic
- And most of the time, more accurate

# Community challenges

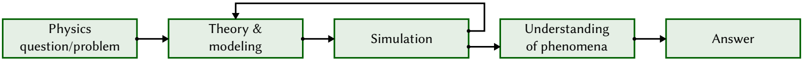
# Simulation: expectation vs reality

Expectation

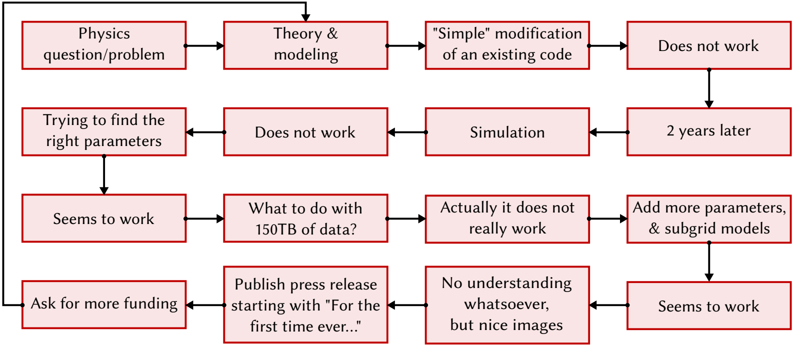


# Simulation: expectation vs reality

## Expectation




## Reality



# Let's talk about computers

## Warning

I am talking about computers

## Simulations are the end result of an interdisciplinary work

- Astrophysics
- Mathematics
- Algorithms
- Data structures
- Parallelism
- Software architecture (can be based on category theory)
- HPC and software/hardware co-design

## Computers are despicable/shameful

- As a matter of fact, most of the community consider the computer side of what people do as "a technical detail" or "engineering stuff nobody cares about"
- Identification of [algorithm], [code], [implementation] and [other computer stuff]

# Let's talk about computers

## A problem of terminology?

Maybe a French problem, one word "informatique", for two concepts:

- IT: Information Technology
- CS: Computer **Science**

## Consequences

- Computer stuff is often considered as an "engineering problem": computer science is ignored
- Working on implementations is often considered as an "academic suicide" as noted during the Exascale Computing in Astrophysics Conference (2013)
- A lot of codes are blocked in the 70's or 90's, ignoring computer science results since then (both fundamental and applied computer science)

## HPC vs observational projects

- Numerical Simulation $\Rightarrow$ Computer Aided Theory
- Big collaborations on telescopes and instruments
- Very few equivalent to develop tools and codes for HPC

# Performance challenges

# Pure computing power vs data transfer

**Data transfer timings**

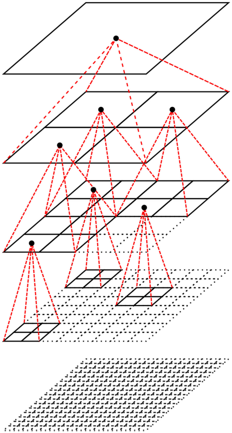
| Operation | Approx. time | Remark |
|---|---|---|
| L1 cache reference | 0.5 ns | |
| One cycle on a 3 GHz processor | 1 ns | |
| Branch mispredict | 5 ns | |
| L2 cache reference | 7 ns | 14× L1 cache |
| Mutex lock or unlock | 25 ns | |
| Main memory reference | 100 ns | 200× L1 cache |
| Send 1 KB over a 1 Gbps network | 10 $\mu$s | |
| Read 1 MB sequentially from main memory | 250 $\mu$s | |
| Round trip within the same datacenter | 500 $\mu$s | |
| Read 1 MB sequentially from a SSD | 1 ms | 4× memory |
| Disk seek | 10 ms | 20× datacenter RT |
| Read 1 MB sequentially from disk | 20 ms | 80× memory |
| Send packet California→Netherlands→California | 150 ms | |

## Consequences

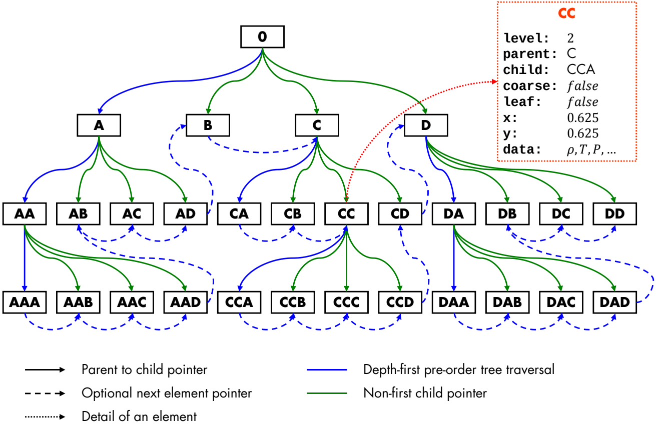
- Most of the time, pure computing time is not the problem
- Most of the time, data transfer is the problem:
  [disk] → [memory] → [cache]
  [cache] ← [node memory] ↔ [node memory] → [cache]
- Once everything is in cache, computations are fast

# Data structures

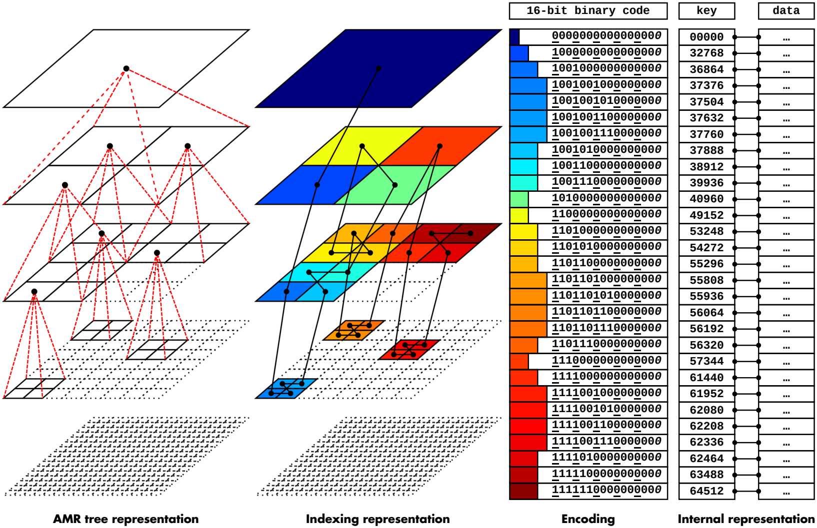
**AMR tree representation**



**Quadtree structure based on pointers**



# Data structures



# Examples of directions of investigation

## Data structures

- Work on graph theory

## Pure performance

- Compile-time computation (metaprogramming)

## Parallelism

- Going beyond MPI and OpenMP models
- Creation of dependency graphs to manage asynchronism

# Programming

# Programming challenges

## Questions

- How to design modular software?
- How to make codes in which changing the physics. . . only require changing the physics
- How to make reliable and maintainable softwares?
- How to obtain both performance and genericity?

## Software architecture

- Extremely complex problem
- Comparable to unification in physics, but on the computer science side
- Can require work in theoretical computer science and/or category theory

But enormous advantages in the long run

# Programming challenges

<table>
<tr><td colspan="4">Applications</td></tr>
<tr><td colspan="4">High level libraries</td></tr>
<tr><td>Wrappers and bindings</td><td>Python</td><td>R</td><td>Java</td></tr>
<tr><td>Optimized libraries</td><td colspan="2">Interpreters (Python, R...)</td><td>Virtual machines (JVM)</td></tr>
<tr><td colspan="4">Compiled, native, low level languages (C, C++...)</td></tr>
<tr><td colspan="4">Compilers, mostly written in C and C++ (GCC, LLVM...)</td></tr>
<tr><td colspan="4">Machine layer, assembly instructions</td></tr>
</table>

## Performances and genericity

- Work on compilers
- Work on DSELs: Domain Specific Embedded Languages

# Illustration

# Context

## Motivation

- Long term physics goal: relativistic cosmological code
- Long term numerical goal: genericity and performance

## Methodology

- Build on lessons from DEUS-FUR
- Start from scratch: no concessions on genericity and performances
- Ignore the problem of implementation time

## Illustration: solving the tree problems in full genericity

- Need: generic AMR and k-D trees (N-dimensions, generic contents, platform specific optimizations)
- Working on the abstract problem: solving the problem of trees: machine learning, geolocalization, abstract syntax trees for compilers, XML trees (web)... AMR and k-D trees
- Explicit, implicit and compressed trees generated at compile-time

# Working on bits

## But...

- Implicit trees require fast bit operations

## How?

- So let's work on bits at the fundamental level
- Exploit assembly cryptographic instructions

## Details

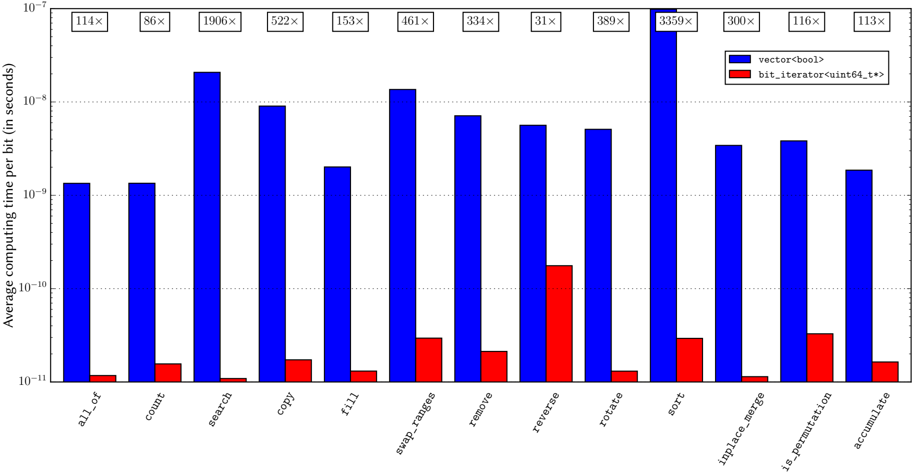
- 6 months of work
- 3 months of paper/pen software architecture
- Applications in cryptography, compression, arbitrary length arithmetic, bioinformatics...
- Will be integrated to the C++ language

# Bit level optimizations

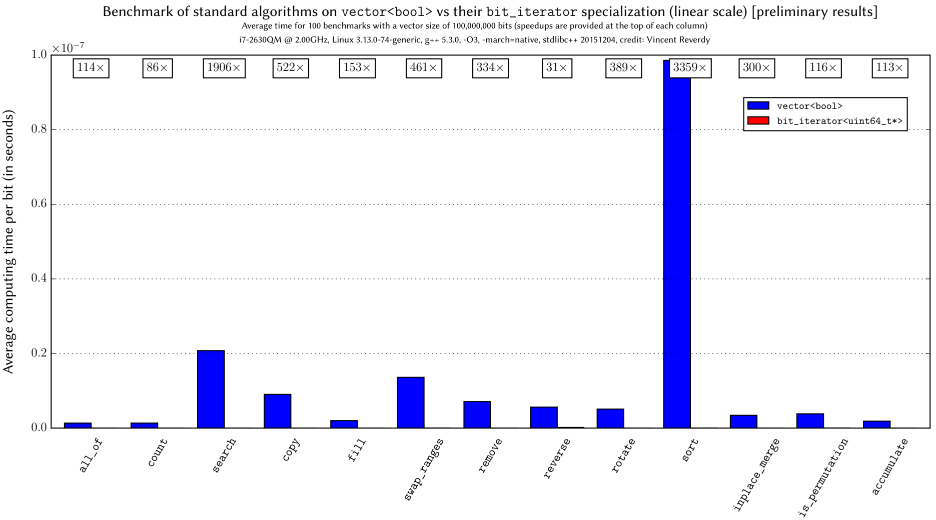
Benchmark of standard algorithms on `vector<bool>` vs their `bit_iterator` specialization (logarithmic scale) [preliminary results]

Average time for 100 benchmarks with a vector size of 100,000,000 bits (speedups are provided at the top of each column)

i7-2630QM @ 2.00GHz, Linux 3.13.0-74-generic, g++ 5.3.0, -O3, -march=native, stdlibc++ 20151204, credit: Vincent Reverdy



# Bit level optimizations



# Conclusions

## Take-home lessons

- Numerical astrophysics is an interdisciplinary field
- More computing power can enable simpler physics exploration
- Exascale simulations will require new approaches
- A lot can be learned from computer science
- Data transfer is a real bottleneck (almost free computation once things are in cache)
- Algorithm and software design are pen/paper exercises
- Not hesitating to solve fundamental problem from scratch can have large benefits

# Thank you for your attention

Any question?