

# ALPSCore tutorial

Alexander Gaenko

Simons Foundation Summer School Workshop  
June 18, 2016

# Overview

This tutorial contains:

- General overview of ALPSCore.
- Exercises:
  - Installation of ALPSCore.
  - Use of ALPSCore to gradually build a feature-rich simulation code.
- The hand-out materials contain:
  - 1 Useful links.
  - 2 Hands-on exercises.
  - 3 Diff files highlighting changes as we put in new features.
  - 4 Diff files that define exercises solutions.

The solutions to the exercises are available online:

[https://git.io/alpstut2\\_solutions](https://git.io/alpstut2_solutions)

# INTRODUCTION

# What it is about

ALPSCore originated from Algorithms and Libraries for Physics Simulations (ALPS) <http://alps.comp-phys.org>

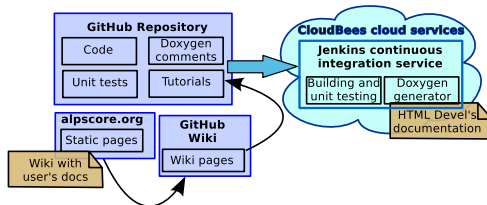
**The grand idea:** Make the [library code](#) from ALPS available with [shorter development cycle](#) and [decent documentation](#).

- What is the [library code](#)?
  - That will most probably useful for many applications.
  - However, ALPSCore is oriented towards Monte Carlo.
- What is the [development cycle](#)?
  - Introducing features (by request?)
  - Testing the features
  - Fixing bugs
  - Documenting
- What is [decent documentation](#)?
  - User's documentations
  - Tutorials (like this one!)
  - Developer's Doxygen-generated reference.

# Web sites

## Contributors:

- Emanuel Gull's group, University of Michigan (USA);
- Lukas Gamper, ETH Zurich (Switzerland);
- ... and many other ALPS contributors.
- Source code: <https://github.com/ALPSCore/ALPSCore>
- Documentation & tutorials: <http://alpscore.org>
- CloudBees for Continuous Delivery.



# Why to use ALPSCore?

Think of a typical MC simulation to-do:

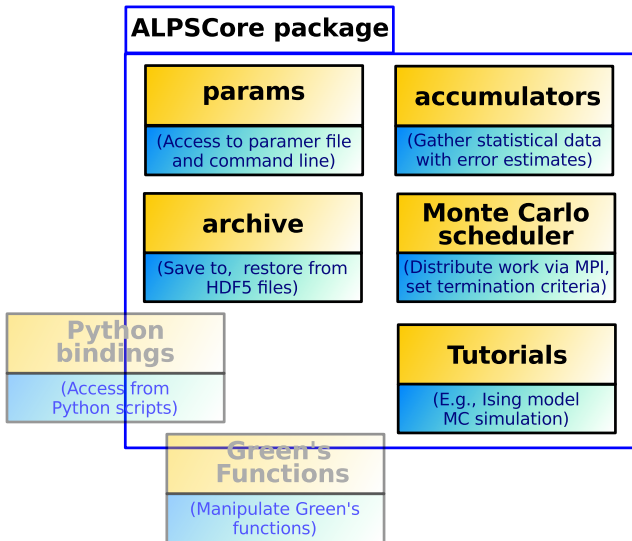
- 1 Read the simulation parameters.
- 2 Decide the step in the phase space.
- 3 Calculate ("measure") the values of interest.
- 4 Collect statistics properly, taking into account autocorrelation.
- 5 Compute derived values with proper error propagation.
- 6 Save intermediate results regularly.
- 7 Set the step or time limit on the simulation.
- 8 Parallelize the whole thing.

Except the **highlighted items**, everything else is "boilerplate".

Sometimes non-trivial one!

Use ALPSCore to minimize boilerplate programming and concentrate on relevant science.

# Overview



# INSTALLING ALPSCore



# Tutorial prerequisites

## What do you need to *know*?

- 1 Basic Unix command-line operations
- 2 Basic C++
- 3 Optional: git
- 4 Optional: installing packages for your OS
- 5 Optional: CMake

## What do you need to *have*?

- 1 Git
- 2 CMake
- 3 C++ compiler
- 4 HDF5 package
- 5 Boost package
- 6 MPI package

# Obtainig ALPSCore

## How to get ALPSCore?

- Downloading release:

`https://github.com/ALPSCore/ALPSCore/releases`

- Getting from GitHub:

`git clone https://github.com/ALPSCore/ALPSCore.git`

- Installing an ALPSCore package:

- Via macports or homebrew on MacOS X.
- Via portage on Gentoo Linux.
- Debian or RedHat binary packages may be created if demand arises.

# Building and compilation

There are (of course!) prerequisites.

- **Tools:**

- C++03 (not C++1x — by policy).  
Tested with GCC 4.2+, Intel 10.0+, Clang 3.2+
- CMake 2.8.12+

- **Libraries:**

- HDF5 1.8+
- Boost 1.54.0+
- MPI (technically, optional)
- GoogleTest (included!)

# Building and compilation

- ALPSCore will try to find Boost, HDF5 and MPI.
- Usually, if you can run `mpicc`, ALPSCore will also find MPI.
- Building using CMake command line:

```
1 $ cmake -DBOOST_ROOT=/path/to/boost \  
2         -DHDF5_ROOT=/path/to/hdf5 \  
3         -DCMAKE_INSTALL_PREFIX=/usr/local/ALPSCore \  
4         /path/to/alpscore/sources  
5 $ make  
6 $ make install
```

- There are some other CMake variables, less frequently used

# Exercise 1

**Exercise 1:** Download and install this tutorial

Open a terminal. Then, enter the following commands.

```
1 $ cd ~  
2 $ mkdir alpstat  
3 $ cd alpstat  
4 $ git clone https://github.com/ALPSCore/Tutorial2.git  
5 $ cd Tutorial2  
6 $ tutorial=$PWD  
7 $ ls -l
```

You should see a list of files and no error messages.

## Exercise 2

### Exercise 2: Download/install prerequisites

#### Ubuntu Linux

```
1 $ sudo apt-get install cmake
2 $ sudo apt-get install libhdf5-dev
3 $ sudo apt-get install libboost-all-dev
4 $ sudo apt-get install mpi-default-dev
```

#### Mac OS X, port system

```
1 $ port install alpscore
```

This will install the latest ALPSCore release (we don't need it!) and prerequisites.

## Exercise 2

### Exercise 2 (cont): Download/install prerequisites

Test that you indeed have them:

```
1 $ cmake --version
2 $ g++ --version
3 $ h5cc --version
4 $ mpicxx --version
```

## Exercise 3

### Exercise 3: Download and install ALPSCore.

```
1 $ git https://github.com/ALPSCore/ALPSCore
2 $ cd ALPSCore
3 $ mkdir build
4 $ cd build
5 $ export ALPSCore_DIR=$PWD/install
6 $ cmake -DCMAKE_INSTALL_PREFIX=$ALPSCore_DIR ..
7 $ make
8 $ make test
9 $ make install
```

- 1: get ALPSCore from GitHub repository.
- 2–4: create a directory for the build
- 5: denote where it will be installed
- 6: generate the build
- 7–9: do the build, run the tests, and install



## USING ALPSCore

# Using ALPSCore in your program

Note: ALPSCore\_DIR is pointing to the installation directory.

- How CMakeLists.txt should look to use ALPSCore:  
[https://git.io/alpstut2\\_s1\\_cmake](https://git.io/alpstut2_s1_cmake)
- Catch: compilers!
- In-source builds are messy

```
1 $ export CXX=$(which needed_cpp)
2 $ export CC=$(which needed_cc)
3 $ mkdir 000build
4 $ cd 000build
5 $ cmake ..
```

# Using ALPSCore in your program

## A few tips and possible catches:

- Most of times, it is enough to remake after any file changes:  
`$ make`
- To speed up on 4 cores:  
`$ make -j4`
- Catch: if CMake files change:  
`$ cmake .`  
in the build directory
- Catch: if compilers change:  
`$ rm -rf CMake*`
- Catch: after accidental in-source build,  
remove the generated files.
- Catch: if ALPSCore itself is updated, regenerate:  
`$ cmake .`  
in your build directory

## Exercise 4

**Exercise 4:** Build and run a dummy program that uses ALPSCore and does nothing.

The code is at [\\$tutorial/step1\\_trivial](#).

CMake file online: [https://git.io/alpstut2\\_s1\\_cmake](https://git.io/alpstut2_s1_cmake)

Source file online: [https://git.io/alpstut2\\_s1\\_main](https://git.io/alpstut2_s1_main)

## Exercise 4

**Exercise 4:** Build and run a dummy program that uses ALPSCore and does nothing.

The code is at [\\$tutorial/step1\\_trivial](#).

CMake file online: [https://git.io/alpstut2\\_s1\\_cmake](https://git.io/alpstut2_s1_cmake)

Source file online: [https://git.io/alpstut2\\_s1\\_main](https://git.io/alpstut2_s1_main)

```
1 $ cd $tutorial/step1_trivial
2 $ mkdir 000build
3 $ cd 000build
4 $ cmake ..
5 $ make
6 $ ./alpsdemo
```

# Parameters

- `alps::params` class is responsible for parameter parsing.
- `boost::program_options` is the engine.
- See Doxygen documentation (link from <http://alpscore.org/>) for detailed info.

## Features:

- One can use input file, override with command line.
- Input file may contain sections: `[title]`
- Parameters must be defined to make it known.
- Unknown parameters are silently ignored.
- Auto-generated help message.
- Accessing an undefined parameter throws an exception.
- You can assign to parameters, which makes them defined.
- Potential information loss  $\Rightarrow$  exception.

## Exercise 5

**Exercise 5:** Build and run a program that uses parameters.

The code is at [\\$tutorial/step2\\_params](#).

Online: [https://git.io/alpstut2\\_s2](https://git.io/alpstut2_s2)

- Play with the different values of parameters.
- Try to override them from the command line.
- Change the program to make “--loud” parameter an integer, with 0 meaning “be quiet”.

## Exercise 5

**Exercise 5:** Build and run a program that uses parameters.

The code is at [\\$tutorial/step2\\_params](#).

Online: [https://git.io/alpstut2\\_s2](https://git.io/alpstut2_s2)

- Play with the different values of parameters.
- Try to override them from the command line.
- Change the program to make “--loud” parameter an integer, with 0 meaning “be quiet”.

```
1 $ cd $tutorial/step2_params
2 $ mkdir 000build
3 $ cd 000build
4 $ cmake ..
5 $ make
6 $ ./alpsdemo
7 $ ./alpsdemo --help
8 $ ./alpsdemo ../params.ini
9 $ ./alpsdemo ../params.ini --count=3
10 ....
```



## Simple simulation class (doing nothing)

- Simulation that just says “I am running.”
- Derived from `alps::mc_base` class.
- Must define virtual methods:
  - `void measure()`
  - `void update()`
  - `double fraction_completed()`
- Should define static method:
  - `static parameters_type& define_parameters(parameters_type&)`
- Passes to the base class constructor:
  - parameters object
  - a PRNG seed offset (will be needed for parallel simulations)

## Exercise 6

**Exercise 6:** Build and run a trivial MC program.

The code is at [\\$tutorial/step3\\_trivial\\_mc](#).

Online: [https://git.io/alpstut2\\_s3](https://git.io/alpstut2_s3)

Note: the simulation code is split into 2 files.

- Build and run.
- Run with small counts.
- Run with large count and small timelimit; time the execution:  

```
$ time -p your_command
```
- Set large time limit and interrupt the program (via Ctrl-C).
- Change `fraction_completed()` so that `--count=0` would mean “till timeout”.
- Change the name of the `update()` method and see it does not compile any more.

## Exercise 6

**Exercise 6:** Build and run a trivial MC program.

The code is at [\\$tutorial/step3\\_trivial\\_mc](#).

Online: [https://git.io/alpstut2\\_s3](https://git.io/alpstut2_s3)

Note: the simulation code is split into 2 files.

- Build and run.
- Run with small counts.
- Run with large count and small timelimit; time the execution:

```
$ time -p your_command
```

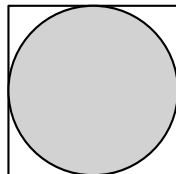
```
1 $ time -p ./alpsdemo --count=10000000 --timelimit=1
```

- Set large time limit and interrupt the program (via Ctrl-C).
- Change `fraction_completed()` so that `--count=0` would mean “till timeout”.
- Change the name of the `update()` method and see it does not compile any more.

# Compute $\pi$ by Markov chain MC

The problem:

- Integral of objective function over an area.
- Trivial Metropolis step to stay inside the area.
- Area: unit square; any step outside is rejected.
- Objective function:  
1 if inside an inscribed circle, 0 otherwise.
- Expected result:  $\pi$  (if multiplied by 4).



We need to use:

- **Accumulators**: named observable to gather statistics
- **Results**: named as accumulators, allow arithmetic operations (with error propagation!)
- Accumulators & Results can hold a vector (e.g., for vector-valued or parametrized objective function)

# Types of accumulators

## Types of accumulators:

- ① Mean only (cheapest, least useful):  
`MeanAccumulator<double>`
- ② No binning (cheap, no autocorrelation info):  
`NoBinningAccumulator<double>`
- ③ Full binning (most expensive, autocorrelation, error propagation):  
`FullBinningAccumulator<double>`
- ④ Log binning (less memory demanding, no error propagation):  
`LogBinningAccumulator<double>`

If a method is not available for the given accumulator type, it throws!

# Exercise 7

**Exercise 7:** Compute  $\pi$  by Markov chain MC.

The code is at [\\$tutorial/step4\\_pi](#).

Online: [https://git.io/alpstat2\\_s4](https://git.io/alpstat2_s4)

- Build and run the program.
- Run with various time limits.
- Run with different step sizes, compare autocorrelation lengths.
- Replace `FullBinningAccumulator` to `NoBinningAccumulator`
- Run with very low or high `--step` and see the underestimated error bars.

# Checkpointing the simulation

- Checkpoint: save intermediate results, load to resume
- ALPSCore utilizes **HDF5 format**
  - Cross-platform
  - Hierarchical structure: Groups ( $\sim$ directories), Data ( $\sim$ files)
- ALPSCore can save/load:
  - Basic types (`int`, `double` *etc.*)
  - Vectors of basic types and of vectors of basic types *etc.*
  - Accumulators and parameters
  - Any user-defined class
    - Via `save()/load()` class members
    - Via traits (harder to do — more complex code)
- Parameters can be constructed from HDF5 file too
  - Should not try to define them again in this case!  
Use `par.is_restored()`.

## Exercise 8

### Exercise 8: Running and resuming.

The code is in `$tutorial/step5_pi_checkpoint`.

Online: [https://git.io/alpstut2\\_s5](https://git.io/alpstut2_s5)

- Build and run the code. There is an error: find and fix it!
- Build, run the corrected code (note more options available!).

```
1 $ ./alpsdemo --help
2 $ # Run for 5 sec
3 $ ./alpsdemo --step 1 --timelimit 5
```

- Note new files appear:
  - “`*.out`” file contains simulation results
  - “`*.clone.h5`” file contains checkpoint
- Restore the checkpoint:

```
1 $ # Run for 10 more sec:
2 $ ./alpsdemo alpsdemo.clone.h5 --timelimit 10
```

- Note:
  - compulsory `--step` is read from the checkpoint
  - parameters can be overridden (like `--timelimit`)



# How to use MPI?

- Not many changes compared to the sequential version.
  - ① Use `alps::mcmadapter<SequentialSimulationClass>` as your simulation class.
  - ② Initialize MPI environment
  - ③ Make sure that the parallel processes do not conflict for input/output
  - ④ Use special constructor for parameters
- Note that the completion is checked only at certain intervals (1 sec minimum)
- Look at the code changes in your handouts.

## Exercise 9

### Exercise 9: Parallel runs.

The code is in `$tutorial/step6_pi_mpi`.

Online: [https://git.io/alpstut2\\_s6](https://git.io/alpstut2_s6)

- Build the MPI-parallelized program.
- Do timed runs with different number of processes.
- Observe checkpoint names.
- Try to restore from checkpoints, see how statistics builds up.

## "Real-world" application: 2D Ising simulation

- The same principle as any other MC simulation:
  - Constructor: generated random spin population.
  - Update step: try to flip a random spin; compute energy change.
  - Measurements: energy, magnetization, magnetization squared.
- Performs arithmetics on results.
- For the sake of simplicity and clarity, a few optimization opportunities missed.
- The program uses a user-defined datatype, therefore needs loading/saving for it.

# Exercise 10

**Exercise 10:** Parallelize the 2D Ising code.

The code is in `$tutorial/step7_ising`.

Online: [https://git.io/alpstut2\\_s7](https://git.io/alpstut2_s7)

Steps:

- 1 Initialize MPI environment.
- 2 Use `alps::mcmpiadapter` template.
- 3 Use the parallel parameter constructor.
- 4 Make sure each rank has its own checkpoint file.
- 5 Make sure only the master process outputs the results.